高等院校信息技术应用型规划教材

Web应用程序开发技术——JSP+Struts 2

李文超　赵新慧　石元博　等　编著

清华大学出版社

高等院校信息技术应用型规划教材

Web应用程序开发技术
——JSP+Struts 2

李文超　赵新慧　石元博 等　编著

清华大学出版社
北京

内 容 简 介

本书由浅入深、循序渐进地介绍了 Java Web 开发技术。本书从 JSP 技术的基础讲起，介绍 JSP 技术的基本语法、内置服务对象、Servlet 和 JavaBean；然后重点介绍基于 MVC 的 JSP 开发框架——Struts 2 技术，通过 Action 组件、拦截器、标签库、结果视图类型、类型转换、输入验证、消息处理、国际化和注解等专题对 Struts 2 框架进行了深入介绍，并将 Web 开发中常用的 JQuery 技术与 Struts 2 相整合。

本书可作为高等院校计算机相关专业的教材，也适合从事 Java Web 开发的初学者使用，还可以作为具有一定经验的 Java Web 开发人员的参考书籍。

图书在版编目（CIP）数据

Web 应用程序开发技术——JSP＋Struts 2/李文超等编著. --北京：清华大学出版社，2013
高等院校信息技术应用型规划教材
ISBN 978-7-302-32021-0

Ⅰ. ①W… Ⅱ. ①李… Ⅲ. ①JAVA 语言－网页制作工具－高等学校－教材 Ⅳ. ①TP312 ②TP393.092

中国版本图书馆 CIP 数据核字(2013)第 078587 号

责任编辑：孟毅新
封面设计：傅瑞学
责任校对：袁　芳
责任印制：宋　林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795764
印 刷 者：三河市君旺印装厂
装 订 者：三河市新茂装订有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：19　　**字　　数**：424 千字
版　　次：2013 年 7 月第 1 版　　**印　　次**：2013 年 7 月第1 次印刷
印　　数：1～3000
定　　价：38.00 元

产品编号：048572-01



基于 Java Web 的应用开发是当前软件开发的主要方向之一，很多高校的计算机类专业都开设了与之相关的课程。作者在多年的授课过程中发现，尽管各出版社出版了大量与该方向相关的书籍，但是这些书籍往往存在两个极端——要么内容过于基础，只是停留在最基本的 JSP 语法和 Servlet 的学习上；要么是内容比较高深，开篇就是基于框架的开发技术，弄得读者晕头转向。

从实际开发的角度上看，已经很少有人使用向 JSP 中插入 Java 代码的方法进行 Web 应用程序的开发了。基于 Java 的 MVC 开发，尤其是基于 Struts 2 的开发是软件公司采用的主流技术。作者编写本书的初衷就是想弥合前面提到的两类教材，使得初学者利用较短的时间学习了基本的 JSP 语法后，逐步过渡到基于 Struts 2 的应用开发中。为此，我们组织了辽宁石油化工大学具有多年教学经验和软件开发经验的教师编写了这本教材。

本书在内容组织上由浅入深、循序渐进，共分 13 章。

第 1 章概述了目前主流的 Web 开发技术和利用 MyEclipse 开发 Java Web 应用程序的方法。

第 2 章和第 3 章讲述了基本的 JSP 技术，包括 JSP 语法、JSP 内置对象、JavaBean、Servlet、JDBC 和 MVC 技术等内容。在讲解这部分内容时，我们摒弃了一些在当前软件开发中过时的知识，并在第 3 章结束时给出了一个留言板程序的开发例程，逐步引导读者从最原始的 JSP 开发方式过渡到基于 MVC 框架的开发。

第 4～12 章详细介绍了与 Struts 2 开发有关的基本技术。其中，第 4 章为 Struts 2 基础，讲解了 Struts 2 应用开发的基本步骤和用户状态跟踪；第 5 章讲解了 Struts 2 的工作原理、Action 配置和 result 映射等内容；第 6 章为标签库，讲解了 Struts 2 提供的各类数据标签、控制标签和表单标签的运用；第 7 章为拦截器，讲解了拦截器的工作过程和使用方法；第 8 章为文件的上传和下载，讲解了文件上传组件，重点介绍文件上传的开发步骤，并详细介绍基于 Struts 2 框架的文件下载应用开发；第 9 章为输入验证，讲解了 Struts 2 框架的内置验证器的使用方法；第 10 章为消息处理与国际化，讲解了资源文件的格式、资源文件的分类及资源文件的加载顺序，重点介绍如何在 Action 类和 JSP 文件中访问资源消息；第 11 章为类型转换，讲解了 Struts 2 框架对类型转换的支持，重点介绍如何处理集合类型转换，并详细介绍自定义类型转换器的开发步骤；第 12 章为注解，介绍

Struts 2 约定和利用注解替代在 struts.xml 中配置 Action、Result 和拦截器的方法。

第 13 章为整合 JQuery，介绍了一个优秀的、开源的 JS 库，重点介绍利用 JQuery 调用 Action，实现 AJAX 的方法。

本书具有如下特点。

(1) 内容较全面，涵盖了 JSP 基础知识和 Struts 2 Web 开发框架的知识，是国内第一本将二者有机结合的教材。

(2) 在讲解基本知识的同时，注重对 Struts 2 框架的剖析，有利于读者了解 Struts 2 实现的机理。

(3) 本书还注重知识的综合运用，采用渐进开发的方法介绍了一个完整案例的设计过程。

(4) 除第 1 章外，本书每章的最后配有一个同步训练。读者通过完成同步训练，增加对本章知识的理解，并训练自己的动手能力。

本书从内容的组织上，适合初学者从零开始学习、进阶，不断深入，循序渐进。本书不仅可以作为大学计算机及相关专业的选修课教材，也适合自学者及网站开发人员参考使用。

本书主要由辽宁石油化工大学的李文超、赵新慧和石元博编写，杨妮妮和冯锡炜等参与了部分章节的编写工作。

在编写过程中，我们参考了大量书籍和文献资料，查阅了大量的博客文章。在此，向本书参考文献的作者表示衷心的感谢，向无私撰写博文、默默耕耘的各位博主表示衷心的感谢，向给予本书帮助的所有人士表示衷心的感谢，尤其感谢沈阳理工大学职业应用技术学院的刘平教授，他为本书的编写提供了很多帮助。

由于编者水平有限，书中难免有不足之处，欢迎同行批评指正。

编　者

2013 年 5 月

目录 CONTENTS

Chapter 1 第1章 Web开发技术概述

1.1 Web技术的发展

随着网络技术的发展，Web技术推陈出新，大致经历了静态文档、动态网页和Web 2.0三个阶段。

1.1.1 静态文档

第一阶段的Web是由静态Web页面构成。每个Web站点都有一个主页作为进入站点的入口，站点中的每个页面都是利用HTML格式编写的，内容相对固定，没有后台数据库，不含程序，不可交互。网页设计人员编写的是什么，它显示的就是什么，不会有任何改变。静态网页中包含大量的超链接，通过超链接允许访问者从一个网页跳转到另一个网页，从一个Web站点跳转到另一个Web站点。Web站点中的每一个静态网页都对应服务器上一个实实在在的文件，都具有一个固定的URL，并且网页的URL以.htm、.html、.shtml等常见形式为后缀，而且不含有“?”。

HTTP(HyperText Transport Protocol，超文本传送协议)是Web服务的通信协议。通过HTTP协议，可以将Web页面从Web服务器传送到Web浏览器。

早期的静态Web页面中只能包含单纯的文本内容，后来逐渐提供了对图片的支持，基本上能够满足建立Web站点的初衷，能够实现对信息资源的共享。然而，随着互联网技术的发展，利用纯粹的静态Web页面展现信息的形式与网上信息几何级增长的矛盾日益突出。人们渴望能够通过将信息存储在后台数据库中，以一种简单的形式，利用少量的Web页面实现对信息的发布和维护。这是静态页面无法实现的。

1.1.2 动态网页

动态网页是为了克服静态页面的不足而引入的一种编程技术。通过在传统的静态页面中加入各种程序和逻辑控制，实现了客户端和服务器端之间的动态和个性化的交流与互动。

需要注意的是，不是在静态网页中增加了Flash动画、滚动字幕等视觉效果之后就形成了动态网页。无论一个网页中是否包含动态效果，只要其内容是通过编程的方式，利用数据库中的数据生成的，这个网页就是动态网页。动态网页的扩展名通常为.jsp、.php、.asp、aspx、.perl和.cgi等形式。

动态网页的页面布局和视觉效果仍然离不开 HTML 的支持，但是由于采用了数据库技术，使得动态网页与 Web 服务器上的文件并不是一一对应的。只有当 Web 服务器接收到用户请求时，才通过运行应用程序将用户所需的内容生成 HTML 格式后，返回给用户。因此，与静态网站相比，使用了动态网页技术的网站的维护工作量大大降低。另外，动态网页提供了交互功能，允许网站实现更多的功能，例如用户权限控制、电子商务和用户留言等。

1.1.3 Web 2.0

Web 2.0 的概念是 2004 年由 O'Reilly 公司提出的，是新的一类互联网应用的统称。Web 1.0 的特点是用户通过浏览器来获取自己感兴趣的信息，主要指的是 Web 发展的第一个阶段，即静态网页阶段。动态网页阶段往往被看做 Web 1.5 时代。Web 2.0 是以 1.0 作为基础设施，添加了一个社交层，注重的是用户的交互作用。在 Web 2.0 中，每个网络用户既是网站内容的浏览者，也是网站内容的创造者。

Web 2.0 的代表网站包括 Facebook 和 Twitter 等。典型的 Web 2.0 应用包括网络社区、网络应用程序、社交网站、博客和 Wiki 等。从开发角度说，Web 2.0 的开发技术包括 AJAX(Asyncronous JavaScript and XML)、提供资料订阅的 RSS(Realy Simply Syndication)、实现内容混播的 Mashup、CSS/XML、P2P(Peer to Peer，点对点通信)和 Flash 等。

简单地说，Web 2.0 使得人们能够更加方便地进行信息的获取、发布、共享，能够以一种更好的形式实现沟通交流和群组讨论。Web 2.0 使得人们成为 Web 社会的人，Web 也有了社会性，成为社会化网络。

1.2 常见应用系统的体系结构

1.2.1 C/S 结构

C/S(Client/Server，客户/服务器)结构是基于资源不对等，且为实现共享而提出来的，也是 2000 年以前最为流行的软件体系结构之一。C/S 体系结构定义了工作站如何与服务器相连，以及如何实现将数据和应用分布到多个处理机上。

如图 1-1 所示，在 C/S 体系结构中，服务器和客户机的地位是不平等的，服务器在硬件配置、运算能力和存储能力上都要优于客户机。因此，服务器构成了网络的核心，网络中的资源主要集中在服务器上，客户机通过访问服务器获得所需要的网络资源。

最简单的 C/S 体系结构的数据库应用由客户应用程序和数据库服务器程序两部分组成。服务器程序在启动后，等待响应客户程序随时发来的请求；客户应用程序实现了用户访问服务器程序的操作接口，当需要对数据库中的数据进行操作时，客户程序自动地寻找服务器程序，并向其发出请求，服务器程序根据预定的规则做出应答，送回结果。

C/S 结构的优点在于可以充分利用服务器和客户机硬件环境的优势，将任务合理分配到客户机和服务器上来实现，降低了系统的通信开销。另外，通过功能划分，客户应用

程序的开发集中于数据的显示和分析。服务器程序的开发则关注于数据的管理，完成对数据库安全性的要求，实现数据访问并发性的控制，以及实现对客户应用程序的全局数据完整性规则等工作。

由于 C/S 结构的系统需要分别开发服务器应用程序和客户应用程序，因此开发成本较高，客户端程序设计复杂，双方交换的信息的内容和形式比较单一，软件的维护和升级困难。

传统的两层 C/S 结构是单一服务器且以局域网为中心的，所以难以扩展至广域网或 Internet。因此，人们提出了三层 C/S 结构，即将应用功能分成表示层、功能层和数据层三部分，如图 1-2 所示。表示层是应用的用户接口部分，位于客户机上，担负着用户与应用间的对话功能。功能层负责接收表示层发来的服务请求，完成具体的业务处理逻辑，由应用服务器完成。数据层就是 DBMS，负责管理对数据库数据的读/写。在有些应用场合，应用服务器和数据库服务器可能在同一台物理服务器上实现。

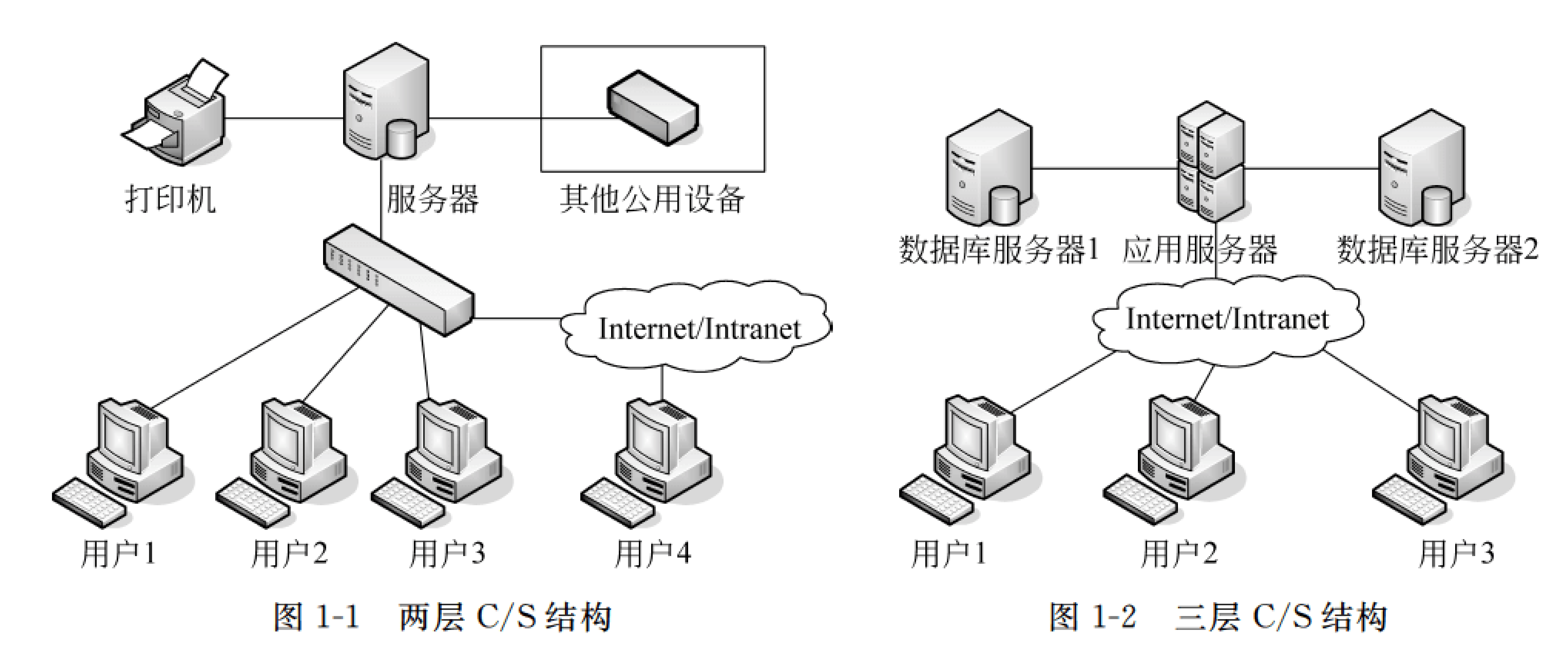


图 1-1　两层 C/S 结构　　图 1-2　三层 C/S 结构

1.2.2　B/S 结构

B/S(Browser/Server，浏览器/服务器)体系结构是由美国微软公司最早提出的，是随着 Internet 的兴起，对三层 C/S 结构的一种变化或者改进的结构，如图 1-3 所示。在这种结构中，Web 浏览器实现了用户界面和一少部分业务逻辑，例如客户端验证等，绝大多数事务功能仍然是在服务器端实现的。

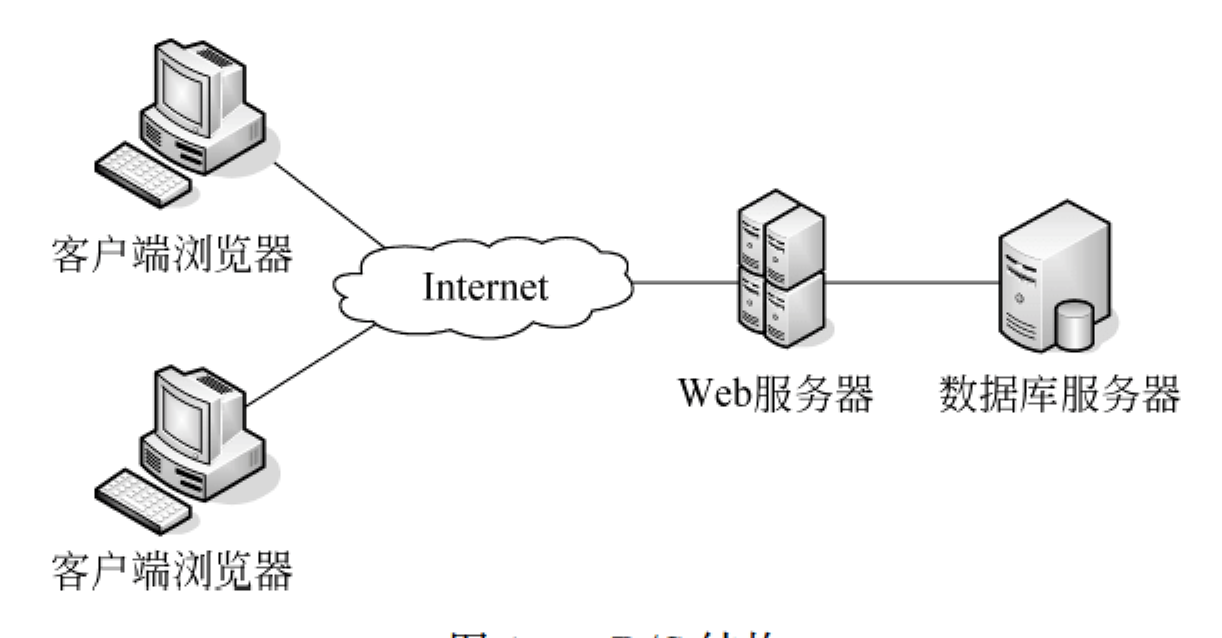


图 1-3　B/S 结构

基于 B/S 体系结构的系统的安装、修改和维护比较简单。由于系统安装在服务器端,用户在使用时,只需要一台浏览器就可以访问系统的全部模块,因此无论客户规模有多大,都不会增加维护工作量,所有的维护工作仅需对服务器进行,真正实现了客户端的零安装和零维护。

当然,与 C/S 结构相比,采用 B/S 体系结构的应用系统在数据查询的响应速度上,要远远低于 C/S 体系结构。安全性问题也是 B/S 结构需要着重考虑的。尽管目前 B/S 结构使得客户端和服务器端能够交换的信息的内容和形式多种多样,但是要想使得用户界面效果达到 C/S 结构的程度,需要程序员付出更多的精力。

另外,目前的 Web 浏览器种类众多,常见的包括 Safari、Google Chrome、Firefox 和 IE 及其衍生产品,这些产品对 W3C 规定的 Web 标准支持程度不同,或者对 Web 标准基础进行了扩充,导致同样的 Web 页面在不同浏览器中的显示效果可能是不同的,因此程序员在进行页面设计时经常要考虑浏览器兼容问题。

1.3 Web 服务器端开发技术

Web 是一种典型的分布式应用架构。Web 应用中的每一次信息交换都要涉及客户端和服务器端两个层面。因此,Web 开发技术大体上分为客户端技术和服务器端技术两大类。

目前流行的服务器端开发技术包括 ASP. NET、PHP 和 JSP 三种。

1.3.1 ASP.NET 技术

ASP. NET 是微软公司推出的新一代构建动态 Web 应用程序的开发平台,是一种建立动态 Web 应用程序的新技术,也是对 ASP 技术的扩展。作为. NET 框架的一部分,ASP. NET 提供了所见即所得的可视化开发方式,Web 页面设计人员可以通过拖拽服务器端控件来建立 Web 页面,程序员可以使用任何. NET 兼容的语言(如 C#、Visual Basic. NET、J#)编写业务逻辑代码。

与 JSP 和 PHP 相比,ASP. NET 的生产效率较高,使用服务器控件,可以在不写或者少写代码的情况下轻松、快捷地创建 ASP. NET 网页和应用程序。另外,由于 ASP. NET 应用程序采用页面与代码相分离的技术,即前台页面代码保存到 ASPX 文件中,后台代码保存到 CS 文件中,当编译程序将代码编译为 DLL 文件后,ASP. NET 在服务器上运行时,可以直接运行编译好的 DLL 文件;并且,ASP. NET 采用缓存机制,提高了运行 ASP. NET 的性能。

ASP. NET 可以有效地集成 Silverlight,能够开发出具有专业图形、音频和视频的 Web 应用程序,增强了用户体验。在 ASP. NET 开发过程中,可以利用 LINQ(Language Integrated Query,语言集成查询)编写 C#或者 Visual Basic 代码,以查询数据库相同的方式操作内存数据,解决信息整合的难度。

由于 ASP. NET 只能运行在微软的平台上,因此在开发和部署 Web 应用程序时,服务器的操作系统通常为 Windows Server。Web 服务器采用微软的 IIS(Internet Information

Service,Internet 信息服务),数据库平台多采用 Microsoft SQL Server。

ASP.NET 最大的缺陷是必须运行于 IIS 之上,这是个曾无数次遭受攻击的系统,由于其臭名昭著的安全性问题,使得很多 IT 专业人士拒绝将他们的网络暴露于 IIS Web 服务器之下。

1.3.2　PHP 技术

PHP(HyperText Preprocessor,超文本预处理语言)是一种服务器端、跨平台、HTML 嵌入式的脚本语言。其独特的语法混合了 C 语言、Java 语言和 Perl 语言的特点,是一种被广泛应用的开源式多用途脚本语言,尤其适合 Web 开发。

采用 PHP 开发 Web 应用程序时,服务器的操作系统一般使用 Linux,Web 服务器可以采用 Apache,数据库多采用平台 MySQL。Linux、Apache 和 MySQL 均为免费、开源软件,PHP+Linux+MySQL 已经成为 Web 开发中的“黄金组合”,这种框架结构可以为网站经营者节省很大一笔开支。

目前在互联网中,有很多网站的开发都是通过 PHP 语言来完成的,例如百度、雅虎、Google、YouTube、Digg 等。在这些知名网站的创作开发中都应用到了 PHP 语言。

但是,PHP 缺乏规模支持以及缺乏多层结构支持,因此不适合应用于大型电子商务站点,而更适合一些小型的商业站点。

1.3.3　JSP 技术

JSP(Java Server Page,Java 服务器页面)技术是一种动态网页技术标准。在传统的网页 HTML,文件中加入 Java 程序片段(Scriptlet)和 JSP 标签,就构成了 JSP 网页。

JSP 具备了 Java 技术的简单易用,完全地面向对象,具有平台无关性且安全可靠,主要面向互联网的所有特点。Java Servlet 是 JSP 的技术基础,早期的大型 Web 应用程序的开发需要 Java Servlet 和 JSP 配合才能完成。

JSP 的优势在于一次编写,到处运行。JSP 具有跨平台特性,几乎可以在所有平台上的任意环境中开发、部署和扩展。JSP 开发的 Web 应用可以运行在 UNIX、Linux 或 Windows 平台上,Web 服务器可以选择 WebLogic 或 WebSphere 等商业平台,也可以使用 Apache 和 Tomcat 等免费的服务器软件,后台数据库可根据实际情况选择 Oracle、Sybase、DB2 或 Informax 等。

与 ASP.NET 比,JSP 入门相对困难,缺少 ASP.NET 中那种只需要简单的数据绑定就可以完成基本任务的服务器控件,许多工作都需要程序员自己编写代码来实现。另外,由于开发工具众多、Web 框架技术层出不穷,JSP 程序员不得不经常面临艰难的选择。

1.4　Web 客户端开发技术

Web 客户端的主要任务是展现信息的内容。为了能够为用户提供一个友好的人机界面,简单地依靠 HTML 是不够的,必须依靠一些客户端开发技术,例如 CSS、JavaScript 和

AJAX 等。下面简单介绍几种在 Web 客户端开发中常用的技术。

1.4.1 CSS

CSS(Cascading Style Sheets,级联样式表)是一种描述性的文本,可以有效地对页面的布局、字体、颜色、背景和其他效果实现更加精确的控制。使用 CSS 允许样式信息(以.CSS 为后缀存储成一个独立的文件)与网页内容(HTML)相分离,使得网站中所有网页的表现风格统一。只要对样式表中相应的代码做一些简单的修改,就可以改变同一页面的不同部分,或者网站的所有网页的外观和格式。

CSS 与 HTML 中的 DIV 标签结合,成为 Web 页面布局设计的最主要技术。

1.4.2 JavaScript

JavaScript 是浏览器能够识别和解释的第一种具有通用目的的、动态的客户端脚本语言。用 JavaScript 可以做许多事情。通过与 CSS 结合,可以使网页更具交互性,能够给站点的用户提供更好、更令人兴奋的体验。例如漂亮的数字钟、有广告效果的跑马灯等。JavaScript 还可以用来完成一部分业务逻辑,以减少服务器的负担,例如表单验证等。

JavaScript 语法简单易学,常用的功能代码在网上随处可见。很多时候,程序员只需要简单地复制、粘贴,就能够实现所需的功能。

另外,一些 JavaScript 库的出现,使得程序员能避开那些乏味的非交互内容,利用库提供的大量方法扩展 Web 页面。利用 JavaScript 库,开发者能够更方便地处理特效、动画、事件、Ajax 及用户交互组件,从而提高开发效率。目前最为流行的 JavaScript 库包括 JQuery、YUI、Prototype、MooTools 和 Dojo。其中的 JQuery 最简洁,也最容易理解,基于 JQuery 的各种插件在网络上随处可见。

1.4.3 XML

XML(Extensible Markup Language,可扩展置标语言)是 W3C 组织给出的一种可扩展的元置标语言,是 SGML 的一个简化子集。这个子集是专为 Web 环境设计的。XML 不像 HTML 那样必须使用已经定义好的标签,而是允许开发人员根据它所提供的规则,制定各种各样的置标语言。严格地说,XML 不应简单地归类到客户端开发技术中。XML 的主要任务在于描述数据,尤其是用来描述结构化的数据。利用 XML,允许在跨平台、分布式或是异质性的环境中以一种中立、标准的格式交换数据。

1.4.4 AJAX

AJAX 是 Asynchronous JavaScript and XML(异步 JavaScript 和 XML)的缩写,它不是一种技术,而是多种功能强大的技术的综合。它使用 XHTML 和 CSS 表达信息;使用 DOM(Document Object Model,文档对象模型)进行动态显示和交互;使用 XML 和 XSLT 进行数据交换和处理;使用 XML HttpRequest 进行异步数据检索;使用 JavaScript 将以上技术融合在一起。

利用 AJAX 技术，可以改善表单验证方式，不再需要打开新页面，也不再需要将整个页面数据提交；不需刷新页面就可改变页面内容，减少用户等待时间；使得客户端能够实现按需获取数据，每次只从服务器端获取需要的数据。利用 AJAX 技术，可以实现客户端异步地与服务器进行交互；在交互过程中，用户无须等待，可继续操作。

AJAX 技术是 Web 2.0 时代最激动人心的技术之一，很多 JavaScript 库中封装了 AJAX 操作，程序员只需要一个简单的方法调用，就能够实现客户端与服务器端无刷新的数据交换。

1.4.5　DOM 和 DHTML

DOM(Document Object Model，文档对象模型)是 W3C 组织推荐的处理可扩展置标语言的标准编程接口。根据该标准，网页中的每一个元素，例如<body>、<div>和<a>都可以直接以可编程的方式访问。DOM 为文档中的元素增加了事件处理能力，使得浏览器能够对用户的输入做出反应。通过执行脚本，可以动态地在网页中增加一幅图片，动态地添加或删除表格中的一行。

DOM、客户端脚本和 CSS 三种技术结合，就是我们平时所说的 DHTML(Dynamic HTML，动态 HTML)。

1.4.6　HTML 5

HTML 5 是近十年来 Web 开发标准最巨大的飞跃。和以前的版本不同，HTML 5 并非仅仅用来表示 Web 内容，它的新使命是将 Web 带入一个成熟的应用平台。在 HTML 5 平台上，视频、音频、图像、动画以及同电脑的交互都被标准化。

HTML 5 强化了 Web 网页的表现性能。HTML 5 提供了很多新的标签，例如，允许使用 JavaScript 直接在网页上绘制图像的 2D 绘图标签<canvas>，这意味着用户可以脱离 Flash 和 Silverlight，直接在浏览器中显示图形或动画；能够完成在线视频、音频播放的多媒体标签<video>和<audio>；完成页面导航和布局的标签<article>、<footer>、<header>、<nav>、<aside>、<section>等。另外，HTML 提供了对本地存储的支持，这个功能将内嵌一个本地的 SQL 数据库，以加速交互式搜索，缓存以及索引功能。一些现有的修饰标签，例如<h1>、<font>等将被剔除，取而代之的是采用 CSS 实现响应的效果。

简而言之，HTML 5 是用于取代 1999 年所制定的 HTML 4.01 和 XHTML 1.0 标准的 HTML 标准版本，尽管目前仍在开发中，但主流的浏览器已经开始支持 HTML 5 的部分元素和部分 API。目前，Internet 上已经有了很多展示 HTML 5 特性的网站，读者不妨去体验一下其激动人心的效果。

1.5　用 MyEclipse 开发 Web 应用程序

MyEclipse 是程序员在开发基于 Java 的 Web 应用时最常用的 IDE 工具之一，下面将介绍如何利用 MyEclipse 开发和部署应用 Web 程序。这里所用 MyEclipse 版本号为 8.6。

1.5.1 创建 MyEclipse Web 项目

首先，启动 MyEclipse，在 File 菜单中选择 New→Web Project 命令，打开 New Web Project 对话框。在 Project Name 文本框中，输入项目名称“Notes”，其他选项可以使用默认的设置，如图 1-4 所示。

图 1-4 New Web Project 对话框

单击 Finish 按钮后，MyEclipse 将创建 Notes 工程的目录结构，包括 src 和 WebRoot 目录，并将 JRE Smstem Library 和 Java EE 5 Libraries 等 JAR 库加到项目的 Build Path 中，如图 1-5 所示。src 目录中通常存放用于处理业务逻辑的 Java 源码，WebRoot 中通常用于存放 HTML、JSP 和图片等静态文件。在实际开发中，程序员会把不同的文件和资源放到不同的目录中。为此，可以通过用鼠标右击 WebRoot，在弹出的快捷菜单中执行 New→Folder 命令来创建所需的子目录。

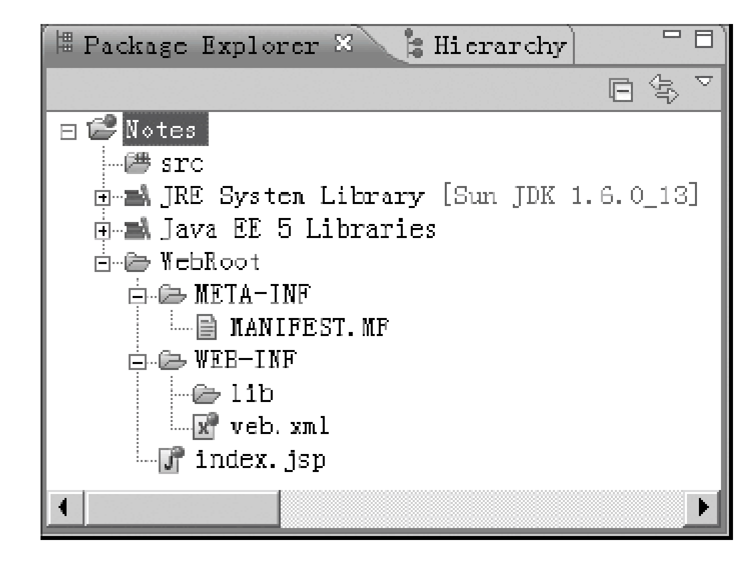


图 1-5 Web 目录结构

WebRoot 下包括一个 WEB-INF 目录和一个站点配置文件 web.xml。通常，在开发项目时用到的 JDBC 驱动、Struts 2 架包等 JAR 文件需要复制到 WEB-INF 目录下的 lib 子目录中，web.xml 中一般包含在项目中用到的过滤器、servlet 和 welcome 文件等的配置。

1.5.2 创建文件

在 Web 应用开发的过程中，经常需要创建 HTML、JSP 和 CSS 等文件。MyEclipse 为常用的类型文件提供了创建向导，程序员可以根据向导的提示，很方便地创建出所需

要的文件。

下面以创建一个 JSP 文件为例来演示 MyEclipse 新建文件向导的用法。首先，在包资源管理器 Package Explorer 中选中新建的 Notes 项目，单击鼠标右键，在弹出的菜单中选择 New→Other 命令，再在选择向导中选择 MyEclipse→Web→JSP，然后单击 Next 按钮。

接下来，在弹出的 JSP 创建向导中，将文件的名字改为 MyJsp. jsp，然后选择使用默认的 JSP 模板，最后单击 Finish 按钮完成创建工作，如图 1-6 所示。

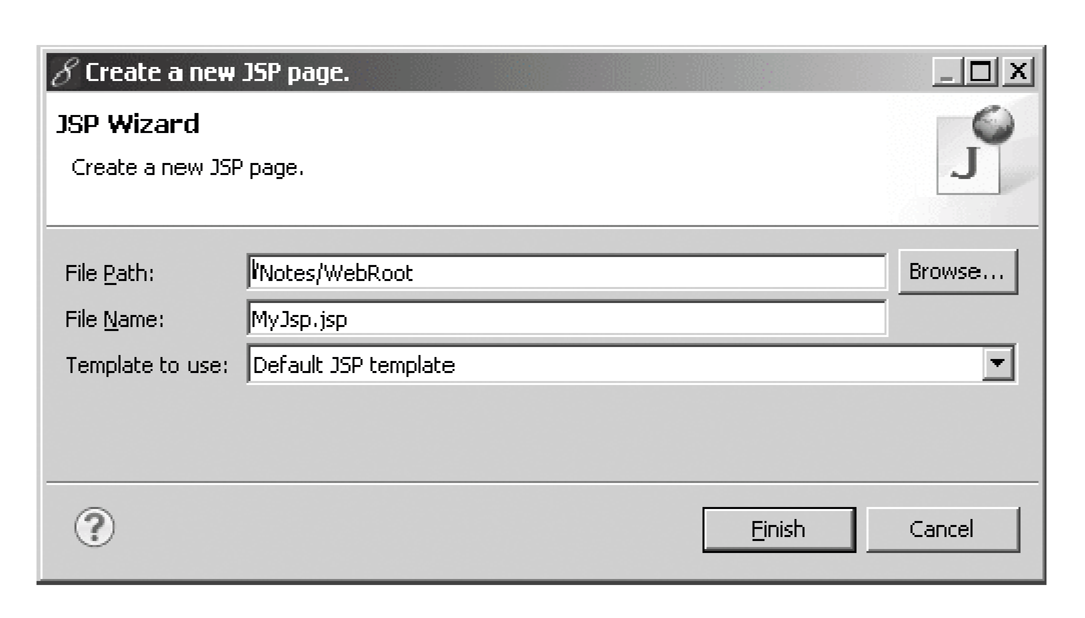


图 1-6　JSP 创建向导

文件创建后，MyEclipse 会自动在编辑器中打开它，程序员可以根据需要来对它进行修改。

1.5.3　配置 Tomcat 应用程序服务器

在 MyEclipse 中能够很方便地将应用程序发布到 Web 服务器上。MyElipse 支持大多数流行的 Web 服务器，Tomcat 就是其中的一个。Tomcat 服务器是一个免费的开放源代码的 Web 应用服务器。Tomcat 是 Apache 软件基金会(Apache Software Foundation)的 Jakarta 项目中的一个核心项目，由 Apache、Sun 和其他一些公司及个人共同开发而成。读者可以到 Tomcat 的官方网站下载最新的版本，网址为 http://tomcat. apache. org。由于 Tomcat 的安装比较简单，在此不再赘述。

下面介绍如何在 MyEclipse 中配置 Tomcat 应用服务器。执行 Window 菜单下的 Preferences 命令，然后在弹出的 Preferences 对话框中依次单击 MyEclipse→Servers→Tomcat→Tomcat 7. x，如图 1-7 所示。选中对话框中的 Enable 选项，接下来利用 Tomcat home directory 后面的 Browse 按钮选择 Tomcat 7. x 的安装路径。有时，需要为 Tomcat 配置 Java 虚拟机，通过单击图 1-7 中左栏 Tomcat 7. x 下的 JDK，切换到 JDK 配置界面添加 Java 虚拟机，如图 1-8 所示。

最后，单击 Preferences 对话框中的 OK 按钮，确认上述操作，完成 Tomcat 服务器的配置。

1.5.4　部署和测试 Web 应用程序

在完成了上面的工作以后，接下来要部署和测试创建的 Web 项目。部署一个 Web 应用程序的过程，就是在 Web 服务器上发布 Web 站点的过程。

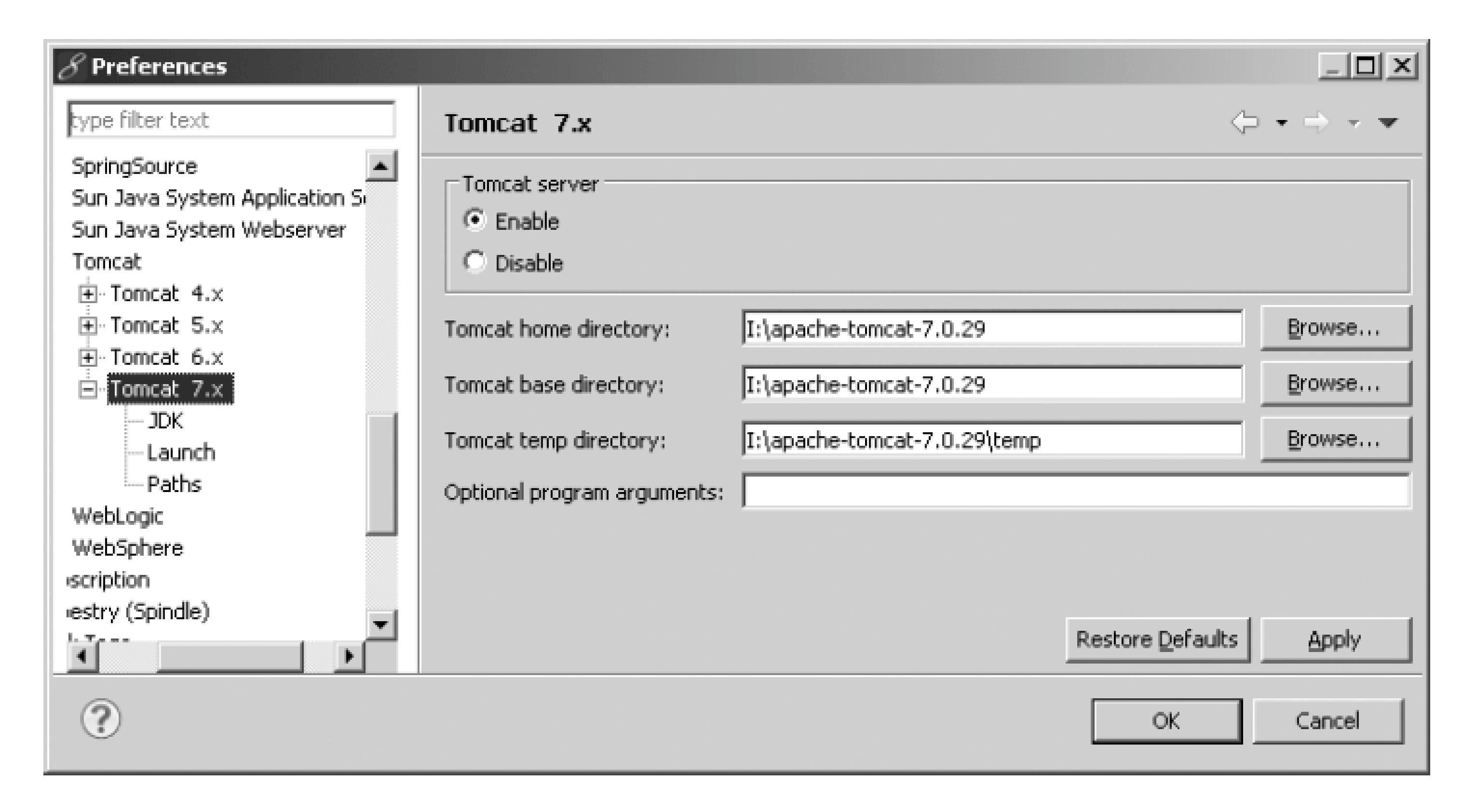


图 1-7 在 MyEclipse 中配置 Tomcat 7. x

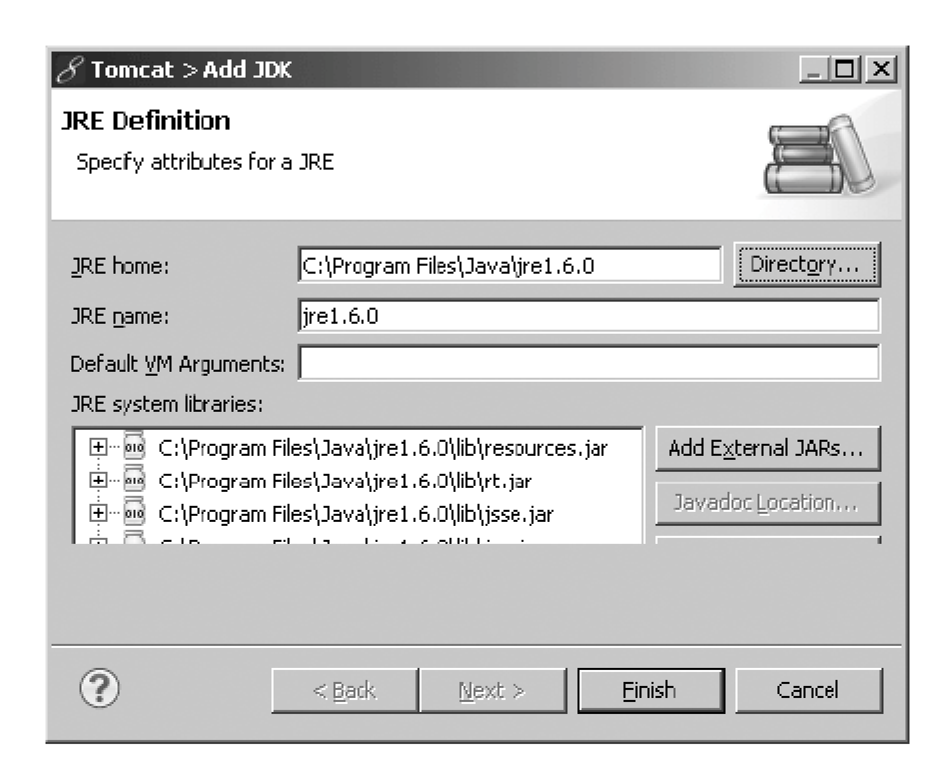


图 1-8 为 Tomcat 配置 Java 虚拟机

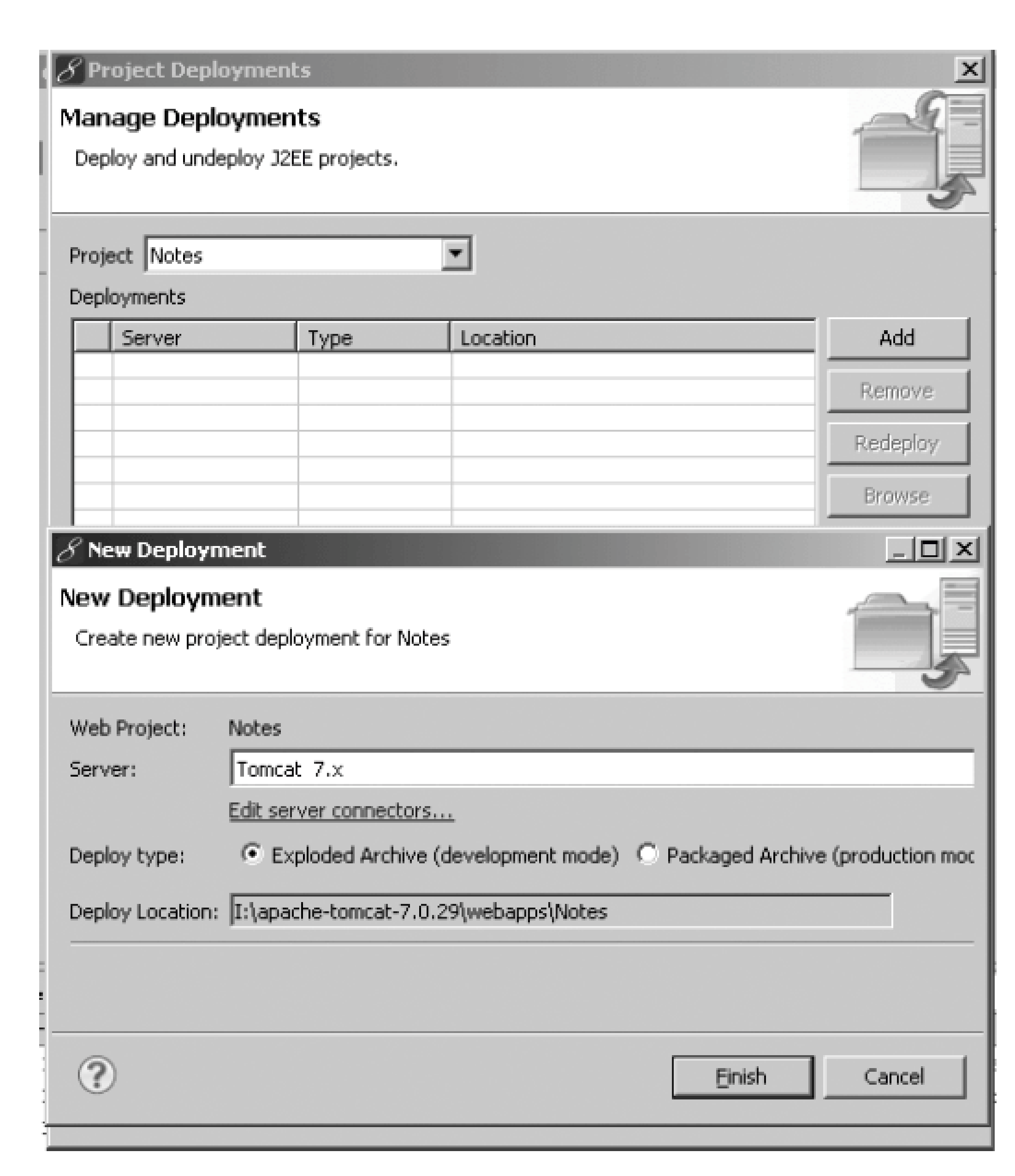


图 1-9 部署 Web 应用程序

单击工具栏中的按钮，弹出 Project Deployments 部署管理对话框，如图 1-9 所示。Project Deployments 对话框中包含三个主要按钮。

(1) Add 按钮：用于创建一个新的 Web 应用程序部署。单击该按钮，在弹出的 New Deployments 对话框的 Server 下拉列表框中选择合适的 Web 服务程序。这里选择 Tomcat 7. x。在 Web 开发阶段，可以选择将 Web 应用程序部署为 Exploded Archive (development mode)，每一个 HTML、JSP 文件都会以独立的形式出现在 Web 服务器中；而在发布成品时，可以选择将 Web 服务程序部署成 Packaged Archive(production mode)，发布到 Web 服务器的所有文件将会被打包到一个 WAR 文件中。

(2) Remove 按钮：用于从 Web 服务器中删除一个 Web 应用程序部署。

(3) Redeploy 按钮：当修改了 Web 服务程序的代码时，可以通过该按钮将改变重新发布到 Web 服务器上。一般情况下，当 HTML、CSS 或者 JSP 文件的内容修改后，MyEclipse 会自动将改变发布到 Web 服务器上；而当 Java 源代码发生修改后，通常需要利用 Redeploy 按钮手工进行重新部署。

成功部署后，单击右边的下拉箭头，然后选择 Tomcat 7. x→Start 启动 Tomcat 应用服务器，根据控制台中的信息查看 Tomcat 是否启动成功。

启动成功后，打开 Web 浏览器，在地址栏中输入 http://localhost:8080/Notes/hello.jsp。如果能看到预期的页面，那么 Web 应用程序部署就成功了。

1.6　对 Web 开发初学者的建议

Web 应用程序设计与传统的桌面程序设计是不同的，初学者首先应该弄清 B/S 的工作原理，弄清服务器和客户端之间的信息交互方式。

学习 Web 开发需要掌握的知识很多，初学者不可能一下子全都学习。建议先熟悉基本的 HTML 标签，能够通过 HTML 标签将需要展示的内容先展示出来。至于页面显示效果，先不必考虑。

从国内目前的开发看，走 Java 路线的开发者，需要学习的知识包括 JSP、Servlet、Struts 2、Spring 和 Hibernate 等。HTML 中嵌入 JSP 标签和 Java 脚本的开发方式已经淡出历史舞台，建议初学者在花费一定的时间弄清基本的 JSP 概念，弄清 Servlet 的工作机制后，将精力投入到基于 MVC 模式的 Web 架构的学习上来。目前来看，Struts 2 是必须要掌握的，尤其是它的值传递功能。至于 Spring，在开发时主要使用的是它的 IoC (Inversion of Control，控制反转)机制。Hibernate 主要用于完成数据的持久化，对于初学者过于复杂。作者更建议从 iBatis 入手。

基本的 CSS 语法必须掌握，不必去背诵每个样式属性的名称，因为利用 CSS 设计工具能够很容易地完成样式的设计。JavaScript 和 AJAX 比较枯燥，Web 前台开发初学者可以在掌握 JavaScript 基本语法和 AJAX 的原理的基础上，直接利用 JQuery 等 JavaScript 库去完成这部分工作。

即使是其中的某一门技术，也是很复杂的。作者在以往的教学过程中，经常发现很多同学捧着厚厚的一本大“砖头”，在很吃力、很细致地“啃”。这是一种费时、自虐式的学习方法，很多人读到一半就很难再读下去了。即使是硬着头皮读下去，等读到最后一页，估计也很难想起前面章节讲授的内容了。作者建议初学者在拿到一本技术书后，能够在一两周内从头至尾看一遍，从全局上了解这门技术都包括哪些方面，每一个方面是解决什么问题的。然后，找一个小的项目，例如一个留言板或者博客程序，自己慢慢摸索着去做，在做的过程中遇到问题了，再带着问题去看书中相应的内容。

“欲善其事，必先利其器”，掌握一个好的开发工具是很重要的。MyEclipse 是进行 Java Web 开发不可缺少的，能够减少开发者的很多工作量。Dreamweaver 是进行页面布

局和 CSS 设计的最佳工具，尤其是 Dreamweaver CS 5.5 更是提供了对 HTML 5 和 JQuery 的支持。Mozilla Firefox 是一个优秀的 Web 浏览器，完全遵循 W3C 标准，在安装上 Firebug 插件之后，开发者能够轻易地对 CSS、HTML、DOM 以及 JavaScript 代码进行调试。

简而言之，要想成为一名 Web 程序开发高手，必须要耐得住寂寞，扎扎实实，一步步地充实自己。

Chapter 2

第2章

JSP 基本语法

2.1 JSP中的HTML代码

2.1.1 HTML常用标签

HTML(HyperText Markup Language,超文本置标语言)是一种用于创建网页的置标语言。利用 HTML 提供的标签可以定义要显示的网页中的各个部分,可以将图片、声音、动画和视频等内容镶嵌到文本文件中。Web 浏览程序通过分析网页文件中的 HTML 标签可以知道如何显示网页信息,如何链接各种信息。

超链接是网页重要的元素之一。各个网页链接在一起后,才能真正构成一个网站。在上网冲浪时,正是利用超链接,用户可以从一个网页很方便地跳转到另一个网页或是另一个网站。

HTML 标签是为了达到某种效果,在内容中加入的特定的标识。标签都括在一对尖括号“＜”和“＞”内,中间加入受标记控制的内容。标签分成单一标签和成对标签两种。单一标签如＜hr/＞和＜br/＞等,成对标签如＜html＞和＜/html＞、＜form＞和＜/form＞等。通常,HTML 标签应该使用小写形式。

下面介绍 HTML 中常见的一些标签。

1. HTML文档结构

利用 HTML 创建的文档称为网页或是 Web 文档。通常,HTML 文件的结构包括文档头部(Head)和文档主体(Body)两大部分。其中,文档头部用于描述浏览器所需的信息,例如网页的标题(Title)和各种 meta 标签等;文档主体包含所要说明的具体内容。代码 2-1 给出了一个简单的 HTML 文档。

代码 2-1　一个简单的 HTML 文档实例

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
        <title>留言板</title>
    </head>
    <body>
        <h1>欢迎使用留言板程序</h1>
    </body>
</html>
```

代码执行结果如图 2-1 所示。



图 2-1　简单的 HTML 文档

meta 标签是可选的，有时利用 meta 标签说明网页所使用的字符集。例如，下面的代码将网页字符集指定为 gb2312(简体中文)：

```
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
```

除 gb2312 之外，比较常见的字符集还有 ISO-8859-1、utf-8 和 GBK 等。

如果希望每隔 5 秒刷新一次网页，可以在文档头部加入以下代码：

```
<meta http-equiv="refresh" content="5"/>
```

其中，http-equiv 和 content 是 meta 标记的属性。大多数 HTML 标记都有各自的属性，在使用标签时可以通过设置这些属性值来获得特定的显示效果。通常，属性值应该使用双引号或者单引号括起来。需要说明的是，现在的设计理念更推崇利用 CSS(Cascading Style Sheet，级联样式表)来设置网页的外观。

2 段落和文字标签

(1) 段落标签<p>和</p>：标签<p>表示一个段落的开始，</p>表示段落的结束。使用段落标签不但可以使文字换行，而且会在不同段落文字间添加一行空白加以间隔。其语法如下：

```
<p>文字</p>
```

(2) 强制换行标签
：在网页中，如果希望换行显示后续内容，一定要使用
，语法如下：

```
文字<br/>
```

(3) 块标签<div>：通常用于实现页面的布局。

(4) 标题标签 hn：类似于 Word 中的一级标题和二级标题等。利用 HTML 中的标题标签，可以定义页面上的一到六级标题。语法如下：

```
<hn>标题文字</hn>
```

其中，n 取值为 1～6。$n=1$ 时，文字最大。

(5) <pre>标签：将在其他文本编辑工具中编辑好的文本粘贴到网页中时，如果希望保留文本的段落格式，可以使用<pre>标签。<pre>标签的一个常见的应用实例就是在网页中发布程序的源码。语法如下：

```
<pre>预定义格式的段落</pre>
```

（6）注释标签：利用注释，可以对 Web 文档的内容添加描述信息。添加了 HTML 注释的内容会和其他内容一样被传送到客户端，但是浏览器在分析网页时会被忽略。

（7）水平线标签<hr/>：在网页中插入一条水平线。语法如下：

```
<hr align=left|center|right size=线粗 width=线宽 color=颜色/>
```

（8）文字标签<font>：用于定义文字的显示外观，语法如下：

```
<font size=数字 face=字体名 color=颜色>文字</font>
```

代码 2-2 为段落和文字标签实例。

代码 2-2　段落和文字标签实例

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>段落标签</title>
    </head>
    <body>
        <h2 align="center"><font color="#FF0000" face="黑体">第 2 章 JSP基本语法</font></
         h2>
        <hr />
        <h3>2.1 JSP中的 HTML代码</h3>
        <h4>2.1.1 HTML常用标签</h4>
        <p>  HTML(HyperText Markup Language,超文本置标语言),是一种用于创建网页的
         置标语言。</p>
        <p>  超链接是网页重要的元素之一。各个网页链接在一起后,才能真正构成
         一个网站。在上网冲浪时,正是利用超链接,我们可以从一个网页很方便地跳转到另一
         个网页或是另一个网站。</p>
        <div>1.强制换行标签 &lt;br&gt;: 在网页中如果希望换行显示后续内容,一定要使用 &lt;
         br&gt;</div>
        <div>2.注释标签:利用注释可以对 Web文档的内容添加描述信息。添加了 HTML注释的内
         容会和其他内容一样被传送到客户端,但是浏览器在分析网页时会被忽略。</div>
    </body>
</html>
```

代码执行结果如图 2-2 所示。

3. 列表标记

HTML 提供的列表中比较常用的是有序列表和无序列表。

（1）有序列表

有序列表标记为<ol>，每一列使用<li>标签定义，每列使用数字或字母开头。格式如下：

```
<ol type="1|A|a|I|i">
    <li>项目 1</li>
    <li>项目 2</li>
    <li>项目 3</li>
```

```
</ol>
```

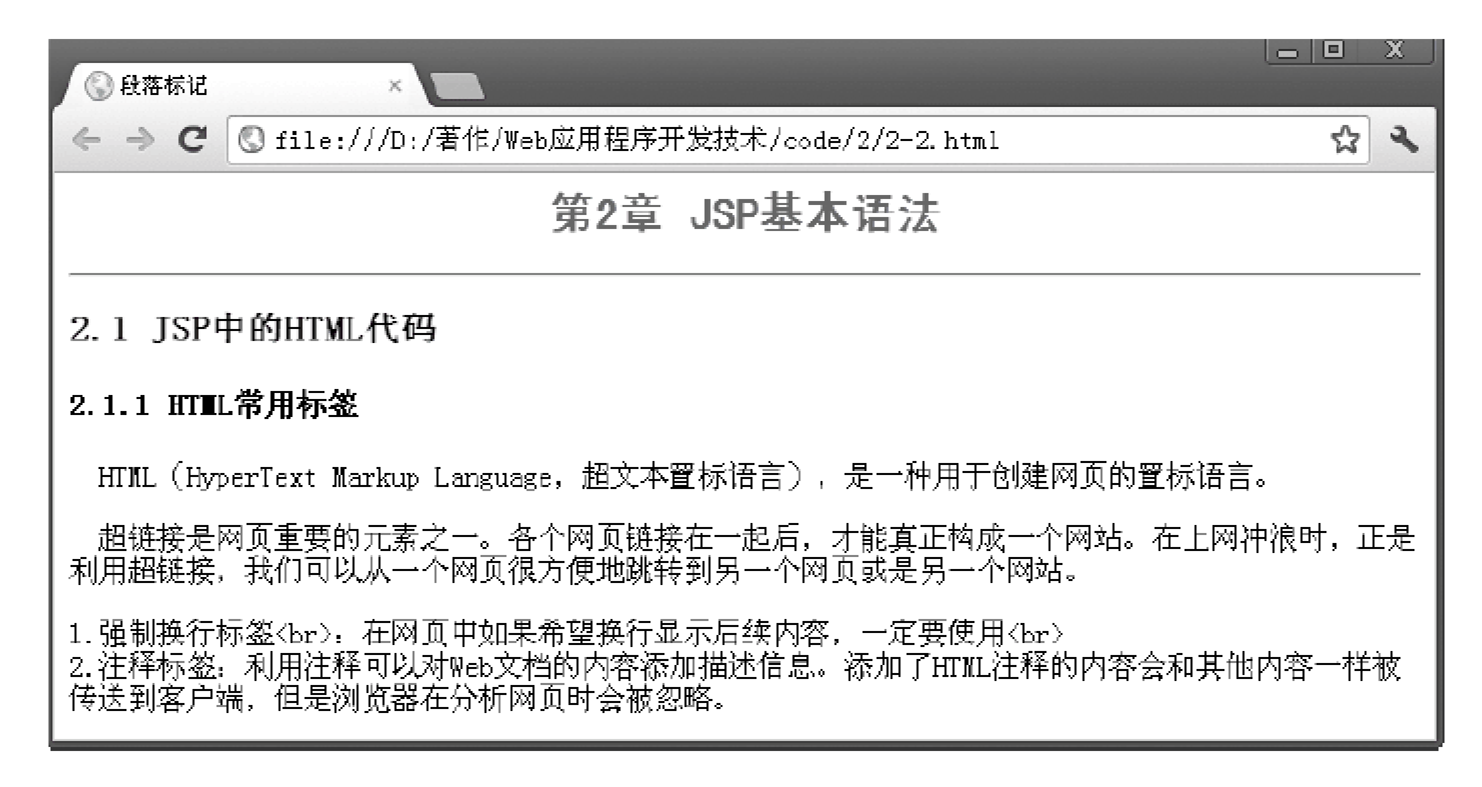


图 2-2 段落和文字标签实例的运行结果

其中，type 属性规定列表的项目符号的类型，默认值为 1，表示列表符号为阿拉伯数字。type 属性还可以取值为 A、a、I、i，分别表示列表符号为大写英文字母、小写英文字母、大写罗马数字和小写罗马数字。

（2）无序列表

无序列表标记为<ul>，格式如下：

```
<ul type="disc|circle|square">
    <li>项目 1</li>
    <li>项目 2</li>
    <li>项目 3</li>
</ul>
```

其中，type 属性规定列表的项目符号的类型，默认值为 disc。

除了这两个常用的列表标签外，HTML 还提供了自定义列表<dl>和选单列表<menu>，请读者自行查阅相关文献。

4. 锚标签<a>

<a>标签可定义锚。锚(anchor)有两种用法。

（1）通过使用 href 属性，创建指向另外一个文档的链接(或超链接)。

（2）通过使用 name 或 id 属性，创建一个文档内部的书签(也就是说，可以创建指向文档片段的链接)。

<a>元素最重要的属性是 href 属性，它指定链接的目标。格式如下：

```
<a href="url">显示的文字</a>
```

5. 表格

表格通常用于文本的结构化显示，由<table>标签来定义。每个表格包括若干行(由<tr>标签定义)，每行包括若干单元格(由<td>标签定义)。数据单元格可以包含文本、图片、列表、段落、表单、水平线和表格等。表格的基本结构如下：

```
<table border="1">
    <tr>
        <td>单元格内容</td>
        …
    </tr>
    …
</table>
```

table 常用的属性包括 border、width、height、align、cellspacing 和 cellpadding 等，这些属性都是可选的。

代码 2-3 给出了一个简单的表格示例。

代码 2-3　表格示例

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  </head>
  <body>
    <table width="465" border="1">
        <tr>
            <th width="54"> </th>
            <th width="134">属性</th>
            <th width="255">说明</th>
        </tr>
        <tr>
            <td rowspan="2">table</td>
            <td>cellpadding</td>
            <td>规定单元边沿与其内容之间的空白</td>
        </tr>
        <tr>
            <td>cellspace</td>
            <td>规定单元格之间的空白 </td>
        </tr>
        <tr>
            <td rowspan="2">td</td>
            <td>colspan</td>
            <td>规定单元格可横跨的列数 </td>
        </tr>
        <tr>
            <td>rowspan</td>
            <td>规定单元格可横跨的行数 </td>
        </tr>
    </table>
  </body>
</html>
```

在代码 2-3 中，表格的首行中包含了三个<th>标签。<th>通常用于定义表格内的表头单元格，其内部的文本通常会呈现为居中的粗体文本。<td>元素内的文本通常是左对齐的普通文本。

图 2-3 给出了代码 2-3 在浏览器中呈现的效果。

2.1.2 HTML 的表单

HTML 表单主要用于采集和提交用户输入的信息。通过 HTML 表单的各种控件,用户可以输入文字信息,或者从选项中选择,以及执行提交的操作。

2-4.html

file:///D:/著作/Web应用程序开发技术/code

	属性	说明
table	cellpadding	规定单元边沿与其内容之间的空白
	cellspace	规定单元格之间的空白
td	colspan	规定单元格可横跨的列数
	rowspan	规定单元格可横跨的行数

图 2-3 表格运行效果

1. 表单标签

表单格式如下:

```
<form action="url" method="post|get">
    <input type="控件类型" name="标识符" />
    ...
</form>
```

其中,各参数的含义如下。

(1) action 属性:用于处理表单请求的文件,可以是一个 JSP 文件、Servlet 类或 Struts 2 的 action 类等。

(2) method 属性:表示了发送表单信息的方式。method 有两个值:get 和 post。默认值 get 的方式是将表单控件的 name/value 信息经过编码之后,通过 URL 发送,在 action 指定的页面返回给用户时,可以从地址栏里看到 name/value。post 方式则将表单的内容通过 http 发送,在地址栏看不到表单的提交信息。从安全性的角度考虑,建议采用 post 方式。

表单中信息的输入主要依靠 input 标签来完成。其中,type 属性定义了输入控件的类型,name 属性和 input 的 value 值构成的 name/value 对在表单提交后交由 action 所指定的文件进行处理。

2. 文字和密码的输入

input 标签的 type 属性为 text 时,输入文本以明文方式显示。input 标签的 type 属性为 password 时,输入文本回显为"*"。

3. 复选框和单选框

复选框格式如下:

```
<input type="checkbox" name="标识符" value="值" checked/>
```

单选框格式如下:

```
<input type="radio" name="标识符" value="值" checked/>
```

checked 表示默认选中该选项。

4. 提交和重置

提交按钮格式如下:

```
<input type="submit" value="按钮名字"/>
```

重置按钮格式如下：

```
<input type="reset" value="按钮名字"/>
```

当用户单击 submit 按钮时，表单的内容会被传送到表单的 action 所指示的文件；当用户单击 reset 按钮时，所有表单中的控件将恢复成默认值。

5. 列表框和下拉框

列表框和下拉框格式如下：

```
<select size=n name="标识符" multiple>
    <option value="值" select>选择项</option>
    …
</select>
```

其中，各参数含义如下。

（1）size：取值为正整数。当 size=1 时，控件外观呈现为下拉框；当 size>1 时，控件外观为列表框，size 值表示列表框中一次可见的列表项数目，如果列表项总数>n，浏览器将自动为列表框添加滚动条。

（2）multiple 表示支持多项选择。

（3）option 中的 select 表示默认选中该项。

6. 多行文本框 textarea

多行文本框 textarea 格式如下：

```
<textarea name="标识符" rows="行数" cols="列数">
</textarea>
```

当多行文本框中的行或列超过 rows 或 cols 指定的值时，浏览器将自动为其添加横向滚动条或纵向滚动条。

代码 2-4 所示为表单运用实例。

代码 2-4　表单运用实例(reg.jsp)

```
<html>
    <head>
        <title>留言板注册</title>
        <meta http-equiv=Content-Type content="text/html; charset=gbk">
    </head>
    <body>
        <form name="regForm"action="doReg.jsp" method="post">
            <p>用  户  名： 
                <input type="text" maxlength="20" size="40" name="username"/>
            </p>
            <p>密     码： 
                <input type="password" maxlength="20" size="40" name="password"/>
            </p>
            <p>重复密码： 
```

```
            <input type="password" maxlength="20" size="40" name="password2"/>
        </p>
        <p>性     别: 
            女<input type="radio" name="gender" value="1"/>
            男<input type="radio" name="gender" value="2" checked />
        </p>
            <p>学     历: 
                <select name="education" size=1>
                    <option value="1">中专</option>
                    <option value="2">大专</option>
                    <option value="3" selected>本科</option>
                    <option value="4">硕士研究生</option>
                    <option value="5">博士研究生</option>
                    <option value="6">其他</option>
                </select>
            </p>
        <p>喜好的运动: 
            <input type="checkbox" name="favorite" value="1" checked/>篮球
            <input type="checkbox" name="favorite" value="2" />足球
            <input type="checkbox" name="favorite" value="3" />爬山
            <input type="checkbox" name="favorite" value="4" />游泳
            <input type="checkbox" name="favorite" value="5" />以上全部
        </p>
        <p>自我介绍:
            <textarea name="intro" rows="10" cols="40"></textarea></p>
        <input type="submit" value="注册"/>
            <input type="reset" value="重置"/>
    </form>
  </body>
</html>
```

代码执行结果如图 2-4 所示。

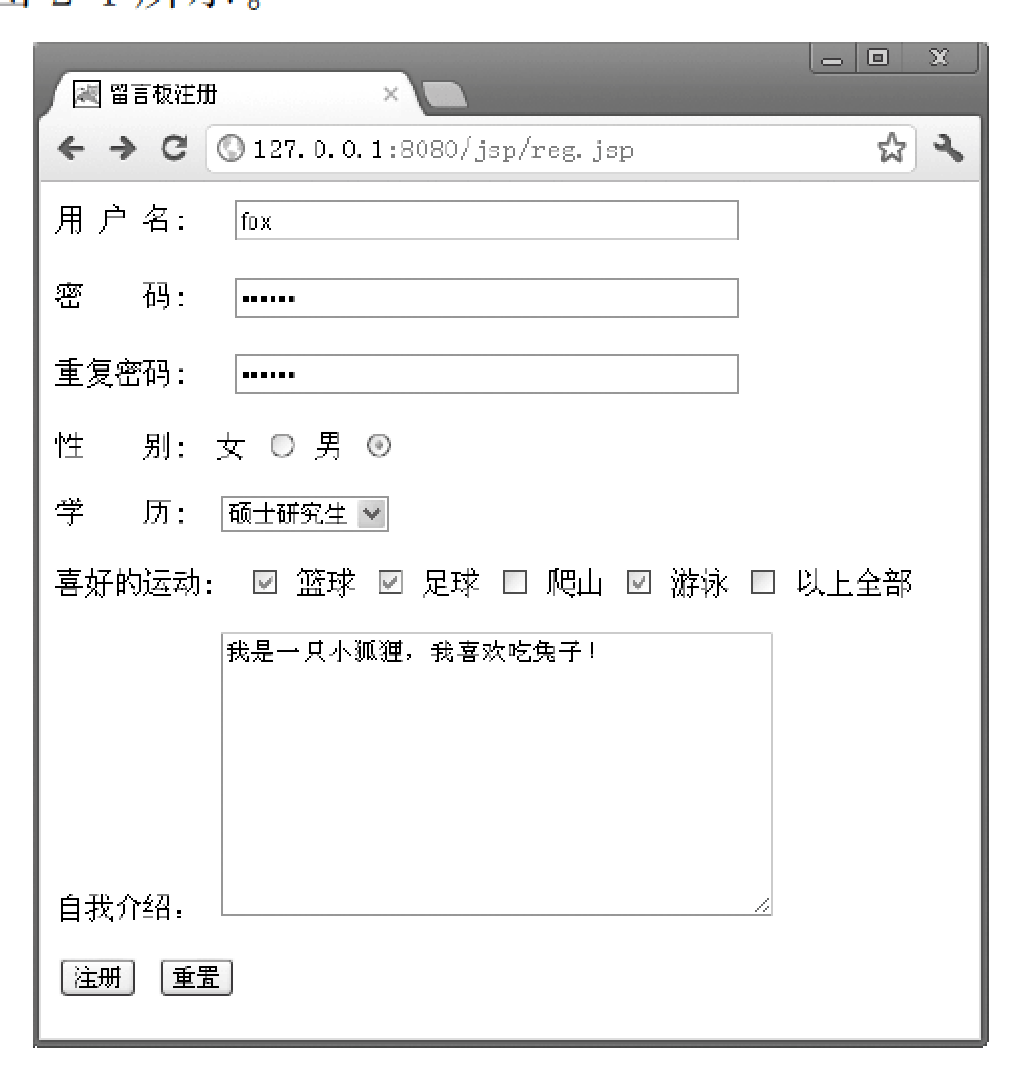


图 2-4　表单运用实例运行结果

2.2　JSP 简介

JSP 是由 Sun Microsystems 公司推出的一种动态网页技术标准。利用 JSP 编写的页面文件的表示层(页面的布局、外观等)仍然采用 HTML 标签完成,业务逻辑则是通过在 HTML 文件中插入 Java 代码程序和 JSP 标签来实现。JSP 将业务逻辑与网页设计和显示分离,支持可重用的基于组件的设计,使得基于 Web 的应用程序的开发变得比较容易。

当客户程序向 Web 服务器请求 JSP 网页时,Web 服务器首先执行 JSP 程序中的程序段,然后将执行结果连同 JSP 文件中的 HTML 代码一起返回给客户。

JSP 具有如下优点。

(1) 跨平台支持。JSP 文件由 HTML 标签和 Java 代码组成,能够广泛地运行在各种平台下。

(2) 编译后执行。第一次访问 JSP 文件时,Web 服务器通过解释该 JSP 页面,生成一个同名的 Java 文件,并将该 Java 文件编译成二进制码(.class 文件),以后再访问时,就直接调用二进制码文件,大大提高执行的效率。

(3) 方便实现业务逻辑与页面表示层相分离。利用 JSP 开发 Web 程序时,通常利用 HTML 或 XML 等设计页面的布局,用 Java 来生成页面中的动态显示内容。通常情况下,Java 代码被封装在 JavaBean 中,在使用 Web 框架的开发中还可能被封装在 Action 或 Servlet 中,使开发人员的分工比较明确,使得 Web 程序维护起来更加方便,页面布局和业务逻辑可以单独修改而不会影响到对方。

4. 支持大型的复杂的企业级应用开发

JSP 完全有能力支持逻辑关系高度复杂的 Web 应用,通过与 Struts 2、Spring、Hibernate 和 Sitemesh 等框架技术结合,能够很好地满足大型的企业级应用的开发。

2.3　JSP 脚本及注释

在传统的 HTML 文件中加入 Java 程序片段和 JSP 标签就构成了一个 JSP 页面文件。一个 JSP 页面可由以下 5 种元素组合而成。

(1) HTML 标签;

(2) JSP 标签,包括指令标签和动作标签等;

(3) 变量和方法的声明;

(4) JSP 的可执行脚本;

(5) JSP 表达式。

2.3.1　JSP 的声明语句

在 JSP 页面中可以定义变量和方法,格式如下:

```
<%! 声明变量或方法 %>
```

在 JSP 页面中，通过声明标识声明的变量和方法，在整个页面内都有效，它们将成为该 JSP 页面被转换为 Java 类后所得到同名类中的属性和方法，并且会被多个线程即多个用户共享。也就是说，其中的任何一个线程对声明的变量或方法的修改都会改变它们原来的状态。它们的生命周期从创建到服务器关闭后结束。

2.3.2 JSP 的可执行脚本

在 JSP 文件中可以通过插入可执行脚本(Scriptlet)来完成指定的业务功能。所谓可执行脚本，就是嵌在“<%...%>”标签中的 Java 代码片段。在脚本中可以定义变量、定义方法、调用方法和进行各种表达式运算。由于可执行脚本本身就是 Java 代码，因此每行语句后面应该添加一个分号。嵌入页面的 Java 代码在 Web 服务器响应请求时会被运行。JSP 的可执行脚本的书写格式如下：

```
<%Java 程序片段 %>
```

需要注意的是，在脚本中定义的变量和方法仅在当前会话的当前页面内有效，不会与其他用户共享。

2.3.3 JSP 的表达式

JSP 表达式用于向页面输出信息，格式如下：

```
<%=表达式 %>
```

注意：

(1) 表达式为任意合法的 Java 语言规范中的表达式。

(2) 表达式末尾不能有分号(“;”)。

(3) JSP 的表达式与在 JSP 页面中嵌入到脚本段中的 out.print()方法实现的功能相同。如果表达式输出的是一个对象，则该对象的 toString()方法被调用。

JSP 表达式主要应用在以下几个方面。

(1) 向页面输出内容，例如：

```
<%String userName="zhangsan"; %>
用户名：<%=userName%>
```

(2) 生成动态的链接地址，例如：

```
<%String path="detail.jsp"; %>
<a href="<%=path%>">详细内容</a>
```

(3) 动态指定 Form 表单处理页面，例如：

```
<%String action="logon.jsp"; %>
<form action="<%=action%>">
...
</form>
```

2.3.4　JSP 的注释语句

注释语句可以帮助程序员识别和理解程序代码。在 JSP 文件中可以使用的注释语句分为三种：HTML 注释、JSP 隐藏注释和脚本注释。

1. HTML 注释

由于 JSP 文件是由 HTML 标签和嵌入的 Java 程序片段构成的，因此 HTML 注释同样适用于 JSP 文件。语法格式如下：

```
<!-- comment [<%=表达式%> | <%JAVA 执行脚本%> ] -->
```

当包含 HTML 注释的 JSP 文件被请求时，JSP 引擎能够识别并执行包含在 HTML 注释中的 JSP 的表达式和可执行脚本。

需要注意的是，HTML 注释会被发送到客户端。

2. JSP 隐藏注释

JSP 隐藏注释的语法格式如下：

```
<%-- 注释内容 --%>
```

与 HTML 注释不同的是，包含在 JSP 隐藏注释中的 JSP 表达式和 Java 脚本不会执行。另外，JSP 隐藏注释中的内容不会发送到客户端，因此这种方式不但可以减少网络流量，安全性也更高。

3. 脚本注释

由于脚本程序中所包含的是一段 Java 代码，所以脚本程序中的注释方法和 Java 中的注释是相同的，包括单行注释、多行注释和提示文档注释三种。读者请自行参阅 Java 相关教材，在此不再赘述。

下面通过代码 2-5 演示 JSP 的脚本及注释实例。

代码 2-5　JSP 的脚本及注释实例

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%!
    /* getWeekOfDate 方法用于计算今天是星期几 */
    String getWeekOfDate(){
        String[] weekDays={"星期日", "星期一", "星期二", "星期三", "星期四","星期五", "星期
                六"};
        Calendar cal=Calendar.getInstance();
        int w=cal.get(Calendar.DAY_OF_WEEK) - 1;
        if (w<0)
            w=0;
        return weekDays[w];
    }
%>
<html>
  <head>
```

```
    <title>JSP基本语法</title>
  </head>
  <!--HTML注释,客户端可以查看到 -->
  <%--JSP注释,客户端不能查看到 --%>
  <body>
    <%if (Calendar.getInstance().get(Calendar.AM_PM)==Calendar.AM) {%>
    上午好!
    <%} else {%>
    下午好!
    <%}%>
    今天是<%=getWeekOfDate() %>
  </body>
</html>
```

代码执行结果及客户端的源代码如图 2-5 所示。

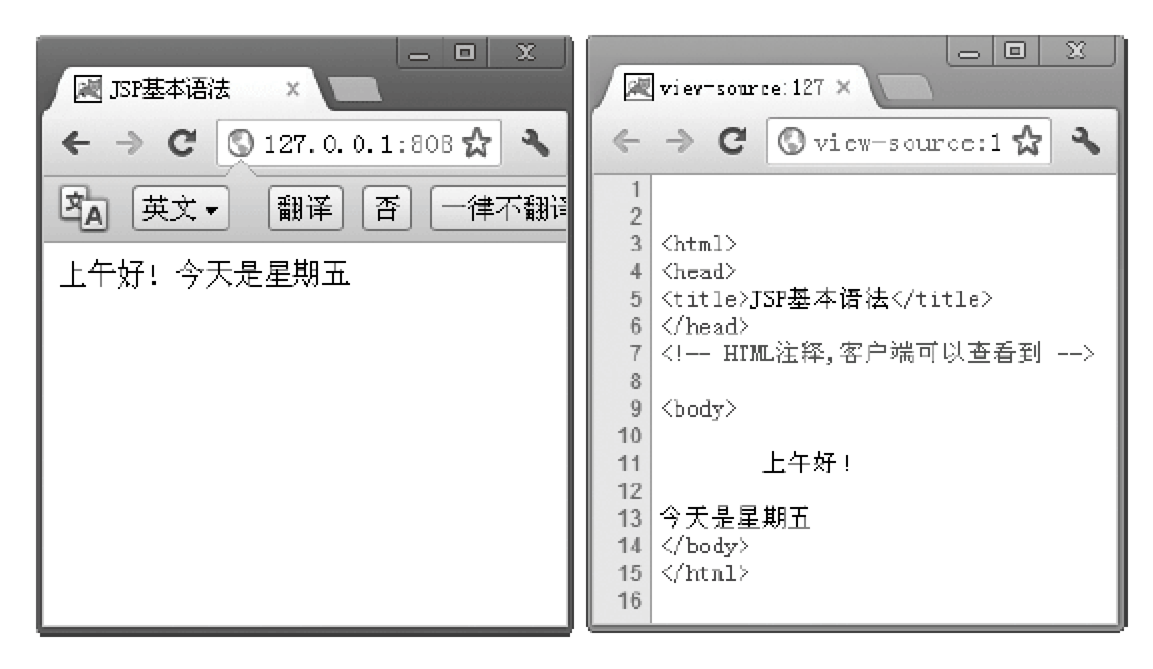


图 2-5 代码 2-5 执行结果及客户端的源代码

2.4 JSP的操作指令

JSP 指令元素主要有三种:page 指令、include 指令及 taglib 指令。每个 JSP 指令都以<%@标签开始,以%>标签结束。三种指令的通用格式如下:

```
<%@指令名称 属性 1="属性值" 属性 2="属性值" ...%>
```

2.4.1 page指令

page 指令用于定义 JSP 文件中有效的属性。格式如下:

```
<%@page
    [language="java"]
    [contentType="mimeType;charset=CHARSET"]
    [import="{package.class|pageage.* },..."]
    [extends="package.class"]
    [session="true|false"]
    [buffer="none|8kb|size kb]
    [autoFlush="true|false"]
```

```
    [isThreadSafe="true|false"]
    [info="text"]
    [errorPage="relativeURL"]
    [isErrorPage="true|false"]
    [isELIgnored="true|false"]
    [pageEncoding="CHARSET"]
%>
```

page 指令可以放在 JSP 页面中的任意位置。page 指令包含多种属性，通过设置这些属性可以影响到当前的 JSP 页面。除 import 属性外，page 指令中的其他属性只能在指令中出现一次。

page 指令各属性的功能如下。

(1) language：设置当前页面中编写 JSP 脚本使用的语言，默认值为 Java。

(2) import：设置当前 JSP 文件中需要导入的类包。在 page 指令中可多次使用该属性来导入多个包。默认情况下，JSP 自动导入包 java.lang.*、javax.servlet.*、javax.servlet.jsp.* 和 javax.servlet.http.*。

(3) contentType：设置响应结果的 MIME 类型。默认的 MIME 类型是 text/html，默认的字符编码为 ISO-8859-1。当多次使用 page 指令时，该属性只在第一次使用时有效。

(4) session：设置当前页面是否支持 session。默认值为 true，表示支持 session。

(5) buffer：设置 out 对象使用的缓冲区的大小。如设置为 none，说明不使用缓存，而直接通过 out 对象进行输出；如果将该属性指定为数值，则输出缓冲区的大小不应小于该值。buffer 的默认值为 8KB。

(6) autoFlush：设置输出流的缓冲区是否自动清除。默认值为 true，说明当缓冲区已满时，自动将其中的内容输出到客户端。如果设置为 false，则当缓冲区中的内容超出其设置的大小时，会产生 JSP Buffer overflow 溢出异常。

(7) isErrorPage：用于说明当前 JSP 页面是否作为错误处理页面。该属性默认值为 false。如果设置为 true，JSP 容器会在当前页面中生成一个 exception 对象，用于捕捉其他页面传回的错误。

(8) errorPage：指定一个当前页面出现异常时所要调用的页面，即 iSErrorPage 属性为 true 的 JSP 页面。

(9) isELIgnored：可以使 JSP 容器忽略表达式语言"${}"。其值只能是 true 或 false。设置为 true，则忽略表达式语言；设置为 false，则不忽略表达式语言 。

(10) pageEncoding：用来设置 JSP 页字符的编码，默认值是 ISO-8859-1。考虑到需要支持中文显示，可以将该值设置为 UTF-8、gb2312 或者 GBK。

2.4.2　include 指令

include 指令用于将另外一个文本文件、HTML 文件或 JSP 文件插入到当前的 JSP 页面中。利用 include 指令可以实现页面的模块化设计，例如在留言板实例中，每个页面的导航部分和版权都是相同的，此时利用 include 指令简化设计过程。由于使用了

include 指令，当需要修改留言板的导航或版权部分时，只需要修改一个 JSP 文件，所有页面的导航或版权部分都会更改，具体用法参见代码 2-6 所示。include 指令的语法格式如下：

```
<%@ include file="被包含文件的 URL" %>
```

代码 2-6 利用 include 指令简化留言板的设计

```
<!-- header.html -->
<html>
    <head></head>
    <body>
        <img src="../images/logo.png"/>
    </body>
</html>

<!-- footer.jsp -->
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>footer</title>
    </head>
    <body>
        <hr/>
        <div align="center">&copy;辽宁石油化工大学 2012 年</div>
    </body>
</html>

<!-- notes.jsp -->
<%@ page language="java"pageEncoding="UTF-8"%>
<html>
    <head>
        <title>留言板</title>
    </head>
    <body>
        <%@ include file="header.html" %>
        <div>
            <ul>
                <li>这是留言板的主体部分,在 notes.jsp 中实现</li>
                <li>导航部分放在 header.html 中</li>
                <li>版权部分放在 footer.jsp 中</li>
            </ul>
        </div>
            <%@ include file="footer.jsp" %>
    </body>
</html>
```

代码执行结果如图 2-6 所示。

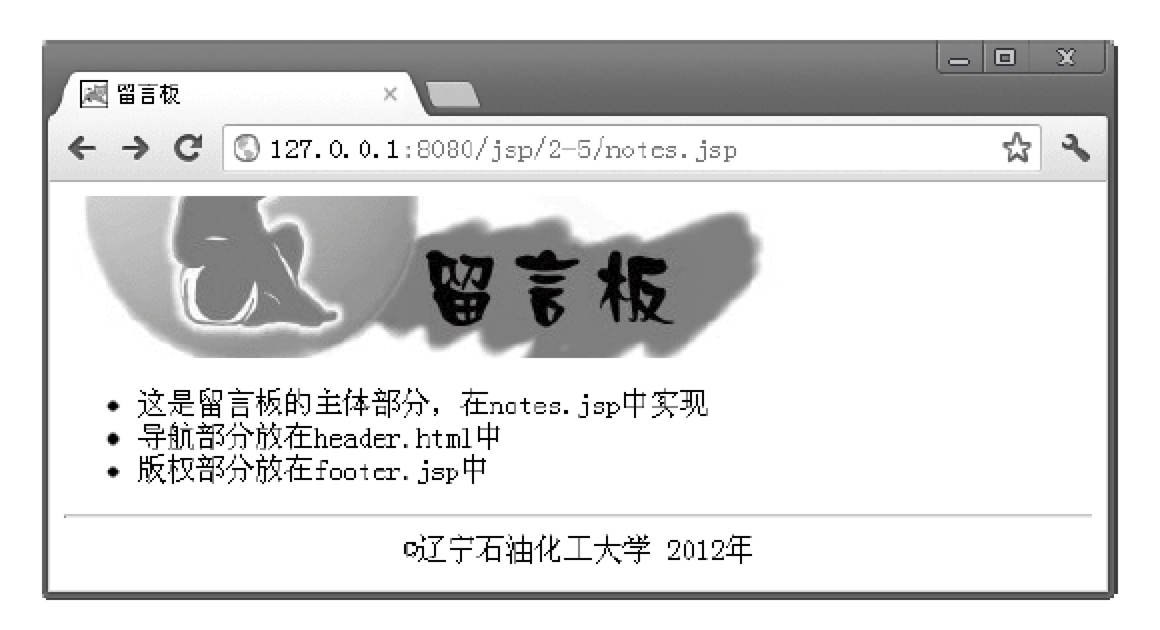


图 2-6　利用 include 指令实现页面模块化设计

2.4.3　taglib 指令

当用户希望在 JSP 页面中使用第三方或自定义的标签库来控制信息的显示时，需要使用 taglib 指令指明标签库的路径和标记前缀。所谓标签库，是一种通过 JavaBean 生成基于 XML 的脚本的方法。标签能够方便地从一个 JSP 项目迁移到其他项目，因此一旦建立了一个标签库，只需要将所有的东西打包为一个 JAR 文件，用户就可以在任何 JSP 项目中重新使用。taglib 的格式如下：

```
<%@ taglib uri="URIToTagLibrary" prefix="tagPrefix" %>
```

其中，uri 属性指定标签库的位置；prefix 属性指定一个在页面中使用该标签库中的标签的前缀。前缀不能命名为 jsp、jspx、java、javax、sun、servlet 和 sunw。

关于利用 taglib 引用标签库的具体用法将在 Struts 2 标签库部分讲述。

2.5　JSP 的动作标签

动作标签是一种特殊的标签，它影响 JSP 运行时的功能。常见的 JSP 动作标签包括<jsp:include>、<jsp:param>、<jsp:forward>、<jsp:plugin>和<jsp:useBean>等。

<jsp:plugin>动作标签用于指示 JSP 加载利用 Applet 编写的插件。由于 Applet 现在使用得越来越少，<jsp:plugin>标签已经很少使用。

<jsp:useBean>动作标签用于加载一个 JavaBean，将在第 3 章学习它。

2.5.1　<jsp:include>动作标签

<jsp:include>动作标签负责把指定文件插入正在生成的页面。其语法如下：

```
<jsp:include page="文件的 URL " flush="true" />
```

<jsp:include>动作标签和前面介绍过的 include 指令的功能有些相似。当 Web 应用程序中各个页面的某些部分（例如标题、页脚和导航栏）都相同的时候，既可以使用 include 指令，也可以使用<jsp:include>动作标签简化页面的设计。不同的是，include

指令采用的是一种静态包含方式，即在 JSP 文件被转换成 Servlet 的时候将被包含的文件插入当前页面；而<jsp:include>动作标签采用的是一种动态包含方式，插入文件的时间是在页面被请求的时候。

因此，当代码 2-6 中的 footer.jsp 发生变化时，只有重新将 notes.jsp 转译成 Java 文件（将该页面重新保存，再访问，就可以产生新的 Java 文件），否则在客户端浏览 notes.jsp 时只能看到修改前的 footer.jsp 内容。如果将语句<%@ include file="footer.jsp" %>修改成<jsp:include page="footer.jsp " flush="true" />，在 footer.jsp 发生变化后，客户端在浏览 notes.jsp 时就会看到 footer.jsp 修改后的内容。

<jsp:include>动作标签引入文件的时间决定了它的效率要稍微差一点，并且被引用文件不能包含某些 JSP 代码（例如不能设置 HTTP 头），但它的灵活性要好得多。

2.5.2 <jsp:forward>动作标签

<jsp:forward>动作标签负责将客户的请求重定向到另外的页面或 Servlet 中。其语法格式如下：

```
<jsp:forward page={"文件的 URL "| "<%=expression%>"} />
```

其中，<jsp:forward>动作标签只有一个属性 page。page 属性包含的是一个相对 URL。page 的值既可以直接给出，也可以在请求的时候通过动态计算获得，例如：

```
<jsp:forward page="/manager/login.jsp" />
<jsp:forward page="<%=someJavaExpression%>" />
```

2.5.3 <jsp:param>动作标签

<jsp:param>动作标签负责传递一个或多个参数到指定的文件中，通常与<jsp:include>、<jsp:forward>和<jsp:plugin>等一起使用，语法格式如下：

```
<jsp:param name="paramName" value="paramValue"/>
```

其中，paramName 表示参数的名称；paramValue 表示参数值。

代码 2-7 给出了两个页面 index.jsp 和 index2.jsp。当客户端访问页面 index.jsp 时，服务器会自动将请求转到页面 index2.jsp，同时将两个参数 username 和 now 的值一起传递给页面 index2.jsp。在页面 index2.jsp 中可以利用 JSP 的内置对象 request 调用 getParameter 方法获取参数的值。

代码 2-7 带有参数传递的 forward 动作

```
<!-- index.jsp -->
<%@ page language="java" import="java.util.* " pageEncoding="utf-8"%>
<html>
    <head><title>index</title>
    </head>
    <body>
```

```
        <%Calendar c=Calendar.getInstance();
            String now=c.getTime().toLocaleString();
        %>
        <jsp:forward page="index2.jsp">
            <jsp:param value="Smith" name="username"/>
            <jsp:param value="<%=now%>" name="now"/>
        </jsp:forward>
    </body>
</html>

<!-- index2.jsp -->
<%@ page language="java" pageEncoding="utf-8"%>
<html>
    <head>
        <title>welcome</title>
    </head>
    <body>
        <%
            String username=request.getParameter("username");
            String now=request.getParameter("now");
        %>
        <h2>    欢迎你,<%=username %>现在时间是:<%=now %></h2>
    </body>
</html>
```

代码 2-7 的运行效果如图 2-7 所示。请读者注意观察图中的 URL 和页面标题。



图 2-7　带有参数传递的 forward 动作运行效果

2.6　JSP 的内置对象

为了方便 Web 应用程序的开发，JSP 提供了 9 种内置对象：request、response、out、session、application、config、pageContext、page 和 exception。这些内置对象在 JSP 页面中可以直接使用，程序员不需要对其实例化。其中，最重要的对象为 request、response 和 session。

2.6.1　out 对象

out 对象是类 javax.servlet.jsp.JspWriter 的实例，主要作用是在 Web 浏览器内输出信息。例如，语句

```
<%out.println("<h1>欢迎使用留言板程序</h1>");%>
```

在运行时将向当前 JSP 页面输出一条文本信息，并以一级标题形式显示。out 对象提供了两个方法向客户端输出内容：print()和 println()。需要注意的是，在 Java 脚本中，out 对象的 println()方法不会输出换行符。要实现换行功能，可以使用以下语句：

```
<%out.println("<h1>欢迎使用留言板程序</h1><br/>");%>
```

在 JSP 页面中，可以通过 out 对象调用 clear()方法清除缓冲区中的内容。这类似于重置响应流，以便重新开始操作。out 对象用于管理响应缓冲区的方法如表 2-1 所示。

表 2-1　out 对象常用的管理响应缓冲区的方法

方　法	说　明
clear()	清空缓冲区
clearBuffer()	清空当前区的内容
close()	先刷新流，然后关闭流
flush()	刷新流
getBufferSize()	以字节为单位返回缓冲区的大小
getRemaining()	返回缓冲区中没有使用的字符的数量
isAutoFlush()	返回布尔值，自动刷新还是在缓冲区溢出时抛出 IOException 异常

2.6.2 request 对象

request 对象是接口 javax. servlet. http. HttpServletRequest 的一个实例，是最重要的服务器对象之一。在搜索引擎或用户注册等程序中，客户端一般通过 HTML 表单或是在 URL 后面添加参数的方法向服务器端提交数据。这些数据被封装在 HTTP 协议报文中，利用 request 对象可以访问 HTTP 头和封装的请求信息。

request 对象最常用的功能是用于获取封装在 HTTP 报文中的请求数据。前面举过一个关于表单的例子(见代码 2-4)，现在编写一个 doReg. jsp 页面完成对表单提交的数据的接收和处理，参见代码 2-8。

代码 2-8　doReg. jsp

```
<%@page language="java" pageEncoding="UTF-8"%>
<%
    //设置 request 字符编码
    request.setCharacterEncoding("utf-8");
    //获取客户请求参数
    String username=request.getParameter("username");
    String password=request.getParameter("password");
    String password2=request.getParameter("password2");
    String gender=request.getParameter("gender");
    String education=request.getParameter("education");
    String[] favorite=request.getParameterValues("favorite");
    String intro=request.getParameter("intro");
```

```
        //本处仅打印出接收到的信息
        out.println("用户名：");
        out.println(username+"<br/>");
        out.println("密码：");
        out.println(password+"<br/>");
        out.println("重新密码：");
        out.println(password2+"<br/>");
        out.println("性别：");
        out.println(gender+"<br/>");
        out.println("学历：");
        out.println(education+"<br/>");
        out.println("最喜爱的运动：");
        for(int i=0;i<favorite.length;i++){
            out.println(favorite[i]+" ");
        }
        out.println("<br/>");
        out.println("简介：");
        out.println(intro+"<br/>");
%>
```

代码 2-8 中的 request. getParameter(String name)用于获得客户请求中的数据。参数 name 与表单中对应控件的 name 属性相同，或者与提交 URL 的参数名相同。如果参数值不存在，返回 null 值。该方法的返回值类型是 String。

当表单中存在多个与 name 相同的控件(例如 checkbox)，或者 URL 中存在多个同名传递参数时，需要使用方法 request. getParameterValues(String name)获取所有的值。该方法的返回值类型是 String[]。

另外，request 对象还提供了一个 getParameterNames()方法。利用该方法可以获得客户端传送给服务器端的所有参数的名字，其结果是一个枚举类型的数据。

注意：*doReg. jsp 中的第 4 行 request. setCharacterEncoding("utf-8")的作用是对客户端请求进行重新编码，用于解决中文乱码问题。*

图 2-8 给出了代码 2-8 的运行结果，其中的数据对应图 2-4 中的数据。

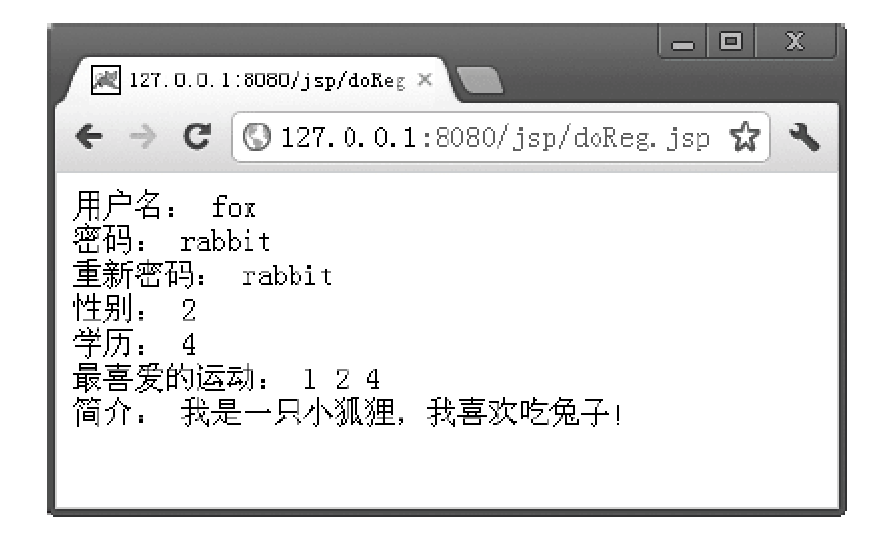


图 2-8　代码 2-8 运行结果

利用 request 对象可以获得客户端的 IP 地址、请求的 URL 等。request 对象常用的方法如表 2-2 所示。

表 2-2　request 对象常用的方法

方　　法	说　　明
getAttributeNames()	返回 request 对象所有属性的名字，返回类型 Enumeration
setAttribute(String name,Object value)	设定名字为 name 的属性，值为 value
getAttribute(String name)	返回 name 指定的属性值。若不存在，返回 null

续表

方　　法	说　　明
getCookies()	返回客户端的 Cookies 对象,返回类型 Cookie[]
getHeader(String name)	获得 HTTP 协议定义的文件头信息
getHeaders(String name)	返回指定名字的 request Header 的所有值,返回一个类型为 Enumeration 的实例
getHeadersNames()	返回所有 request Header 的名字,返回一个类型为 Enumeration 的实例
getMethod()	获得客户端向服务器端传送数据的方法,如 get、post 和 header
getProtocol()	获得客户端向服务器端传送数据所依据的协议名称
getRequestURI()	获得发出请求字符串的客户端地址
getRealPath()	返回当前请求文件的绝对路径
getRemoteAddr()	获取客户端的 IP 地址
getRemoteHost()	获取客户端的机器名称
getServerName()	获取服务器的名字
getServerPath()	获取客户端所请求的脚本文件的文件路径
getServerPort()	获取服务器的端口号

代码 2-9 利用 request 对象获取了客户端和服务器端信息。

代码 2-9　获得客户端和服务器端信息

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>客户端和服务器端信息</title>
    </head>
    <body>
        <h3>客户端信息</h3>
            客户端发出请求所用协议:<%=request.getProtocol()%><br/>
            客户端计算机的 IP 地址:<%=request.getRemoteAddr()%><br/>
            客户端计算机的主机名:<%=request.getRemoteHost()%><br/>
            客户端计算机使用者:<%=request.getRemoteUser()%><br/>
            客户端计算机请求体的 MIME 类型:<%=request.getContentType()%><br/>
            客户端计算机请求所用的设置:<%=request.getScheme()%><br/>
            客户端计算机请求所用的方法:<%=request.getMethod()%><br/>
            客户端计算机请求中包含的字符串:
                                    <%=request.getQueryString()%><br/>
            客户端计算机请求的 URI:<%=request.getRequestURI()%><br/>
            客户端请求中包含的头信息:
        <%
            Enumeration e=request.getHeaderNames();
            while(e.hasMoreElements()){
```

```
                String headerName= e.nextElement().toString();
                out.println("<br>"+ headerName+ ":");
                out.println(request.getHeader(headerName));
            }
        %>
        <h3>服务器端信息</h3>
        <%
            String path= request.getContextPath();
            String basePath= request.getScheme()+ "://"+ request.getServerName()+
                            ":"+ request.getServerPort()+ path+ "/";
        %>
            服务器端的环境路径:<%=path %><br/>
            当前执行的 JSP 的路径信息:<%= request.getServletPath() %><br/>
            服务器端的根路径 URI:<%=basePath %>
    </body>
</html>
```

图 2-9 给出了代码 2-9 的运行结果。



图 2-9　代码 2-9 运行结果

2.6.3　response 对象

response 对象封装服务器处理数据后产生的结果，并将其传回到客户端，用于响应客户的请求。response 对象是接口 javax.servlet.http.HttpServletResponse 的一个实例。利用 response 对象，可以设置 HTTP 标题，设置响应内容的类型和状态，发送 HTTP 重定向和编码 URL 等。

response 对象的常用方法如表 2-3 所示。

表 2-3 response 对象的常用方法

方 法	说 明
addCookie(Cookie cookie)	添加一个 Cookie 对象,用于在客户端保存特定的信息
addHeader(String name,String value)	添加 HTTP 头信息,该 Header 信息将发送到客户端
containsHeader(String name)	判断指定名字的 HTTP 文件头是否存在
sendError(int)	向客户端发送错误的信息
sendRedirect(String url)	重定向 JSP 文件
setContentType(String contentType)	设置 MIME 类型与编码方式

代码 2-10 给出了一个留言板程序的登录页面,用户在输入完用户名 username 和密码 password 后,通过登录按钮将数据发送给 doLogin.jsp 进行处理。

代码 2-10 留言板登录程序

```
<!-- login.jsp 文件源码 -->
<%@ page contentType="text/html; charset=utf-8" language="java"%>
<html>
    <head>
        <title>登录</title>
    </head>
    <body>
        <form action="doLogin.jsp" method="post">
            用户名:<input type="text" name="username" /><br />
            密   码:<input type="password" name="password" /><br />
            <input type="submit" value="登录" />
        </form>
    </body>
</html>

<!-- doLogin.jsp 文件源码 -->
<%@ page language="java" pageEncoding="UTF-8"%>
<%
    //设置 request 字符编码
    request.setCharacterEncoding("utf-8");
    //获取客户请求参数
    String username=request.getParameter("username");
    String password=request.getParameter("password");
    if (username.equals("张三") && password.equals("123")){
        //将用户名保存在 session 中
        session.setAttribute("username",username);
        //重定向到 index.jsp 页面
        response.sendRedirect("index.jsp");
    }
    else
        response.sendRedirect("login.jsp");
%>
```

很多读者都访问过基于网页的聊天室。为了显示最新的聊天内容，聊天室程序需要定时刷新页面内容。这一功能可以通过 response 对象设置 HTTP 头信息来完成。

代码 2-11　response 对象设置 HTTP 头信息

```
<%@ page language="java" import="java.util.Date" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>response 刷新页面</title>
    </head>
    <body>
        <b>当前时间为：</b>
        <%
            response.setHeader("refresh","20");
            out.println(new Date());
        %>
    </body>
</html>
```

代码 2-11 实现了定时刷新页面，每隔 20 秒钟客户端重新发送获取页面请求。程序运行效果如图 2-10 和图 2-11 所示。



图 2-10　代码 2-11 的运行结果

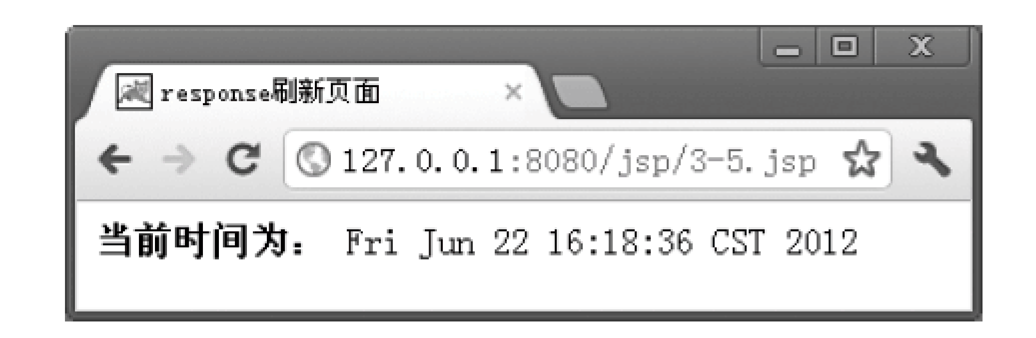


图 2-11　20 秒后更新的结果

2.6.4　session 对象

session 对象用于存储特定的用户会话所需的信息。用户从到达 Web 服务器的某个特定的 Web 页开始，到该用户离开 Web 站点的整个过程称为一次会话。

引入 session 对象是为了弥补 HTTP 协议的不足。HTTP 协议是一种无状态的协议。无状态是指协议对于事务处理没有记忆能力，这种无状态意味着如果后续处理需要前面操作产生的数据，必须将该数据在服务器和客户端之间重传。为了保持 HTTP 之间的连接状态，产生了两种交互技术：cookie 和 session。cookie 是通过客户端保存状态的方案，session 是通过服务器来保持状态的解决方案。session 对象可以让用户在一个 Web 站点的多个页面之间共享少量的信息。在实际应用中，session 对象经常被用于存储登录用户的信息以及实现购物车等。

需要注意的是，JSP 容器会为每个会话用户都设立一个独立的 session 对象，用以存储 session 变量。各个用户的 session 对象互不干扰。

session 对象是 javax. servlet. Http. HttpSession 接口的一个实例，常用的方法如表 2-4 所示。

表 2-4 session 对象的常用方法

方　法	说　明
setAttribute(String name,Object value)	设定名字为 name 的属性,值为 value
getAttribute(String name)	获得指定名字的属性。如果该属性不存在,则返回 null
removeAttribute(String name)	删除名为 name 的属性
invalidate()	销毁 session 对象
isNew()	判断当前 session 是否为新的 session。若是,返回 true;否则,返回 false
setMaxInactiveInterval(int interval)	设置 session 对象的有效时间(也称作最长不活动时间、超时时间),时间单位为秒。所谓有效时间,是指连续两次客户请求之间的最长时间,超过这个时间,session 将失效
getMaxInactiveInterval()	获取 session 对象的最长不活动时间,时间单位为秒

在代码 2-10 中,doLogin.jsp 文件第 10 行代码演示了如何利用 session 保存用户登录信息。代码 2-12 演示了利用 session 保存的信息进行登录验证。

代码 2-12 留言板首页(index.jsp)

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%
    String username=null;
    if(null!=session.getAttribute("username"))
        username=session.getAttribute("username").toString();
%>
<html>
    <head>
        <title>留言板程序</title>
    </head>
    <body>
        <%if(null==username) {%>
            您尚未<a href="login.jsp">登录</a>!
        <%}else{ %>
            欢迎<font color="red"><%=username %></font>使用留言板程序
        <%} %>
    </body>
</html>
```

代码 2-10 和代码 2-12 共同完成了留言板程序的登录部分。index.jsp 页面对用户进行登录验证,未登录用户看到的是“您尚未登录”字样。一旦用户登录后,无论是从其他页面转回到 inex.jsp,还是利用刷新按钮刷新 index.jsp 页面,都将看到“欢迎××使用留言板程序”字样,其中的“××”为当前登录的用户名。

提醒一下读者,session 对象在以下两种情况下失效。

(1) 超过 session 对象的有效期;

(2) 显式调用 invalidate()方法。

session 的有效期也可以在 web.xml 中配置。例如,下面的语句将 session 有效期设置为 60 分钟。

```
<session-config>
    <session-timeout>60</session-timeout>
</session-config>
```

下面讲述 JSP 中 session 实现的原理。

每当服务器接收到一个客户端请求时，首先检查该客户端的请求里是否包含一个 session 标识(JSESSIONID)(见图 2-12)。如果包含一个 session ID，说明以前为此客户端创建过 session，于是服务器从内存中按照 session ID 检索出 session(如果检索不到，可能会新建一个)；如果客户端请求中不包含 session ID，服务器会认为当前客户端发起了新的会话，会为该客户端创建新的 session 并生成与此 session 相关联的 session ID。这个 session ID 将在本次响应中返回给客户端保存。以后，客户端将利用该 session ID 作为会话的凭证，即使浏览器被关闭，只要使用某种手段改写浏览器发出的 HTTP 请求头，把原来的 session ID 发送给服务器，再次打开浏览器时，仍然能够找到原来的 session。

Request Headers	
Accept	text/html, application/xhtml+xml, application/xml;q=0.9, */*;q=0.8
Accept-Encoding	gzip, deflate
Accept-Language	en-us, en;q=0.5
Connection	keep-alive
Cookie	JSESSIONID=0ABE4E6C85BD465620DDD9BFAC71DCCA
Host	127.0.0.1:8080
Referer	http://127.0.0.1:8080/Login/servlet/LoginServlet
User-Agent	Mozilla/5.0 (Windows NT 5.1; rv:13.0) Gecko/20100101 Firefox/13.0.1

图 2-12 Request 头部中包含的 JSESSIONID

2.6.5 application 对象

与 session 对象类似，application 对象也用于存储和访问来自任何页面的变量。不同之处在于，session 对象和用户的关系是一一对应的，用来记录用户的个人信息；application 对象则被所有的用户共享，用来记录全局的信息。

与 session 对象类似，application 对象也可以保存属性和属性的值。session 对象中保存的属性只在用户当前会话范围内有效，一旦会话结束或者会话超过最大不活动时间，session 对象将被销毁；而在 application 对象中保存的属性在整个应用程序范围内是有效的，即使所有的用户都不发送请求，只要不关闭或重启应用服务器，在其中保存的属性都是有效的。

application 对象是 javax.servlet.ServletContext 接口的一个实例，常用的方法如表 2-5 所示。

表 2-5 application 对象的常用方法

方　法	说　明
setAttribute(String name, Object value)	在 application 对象中保存一个名字为 name 的属性，值为 value
getAttribute(String name)	获取指定的属性值。如果该属性不存在，则返回 null
getAttributeNames()	获取所有可用属性名，返回类型为一个 Enumeration 类型实例
removeAttribute(String name)	从 application 中删除名为 name 的属性
ServletContext getContext(String uripath)	获取指定 URL 的 ServletContext 对象

下面通过一个网页访问次数计数器的例子来说明 application 对象的使用方法，如代码 2-13 所示。

代码 2-13　网页访问计数器(applicationCount.jsp)

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%
    int count=0;
    //从 application 对象中取得已经访问次数 count
    Object o=application.getAttribute("count");
    if(null!=o)
        count=Integer.parseInt(o.toString());
    //访问次数加 1
    count++;
    //将访问次数保存到 application 对象中
    application.setAttribute("count",String.valueOf(count));
%>
<html>
    <head>
        <title>网页访问计数器</title>
    </head>
    <body>
        当前网页的访问次数为：<%=count %>
    </body>
</html>
```

代码 2-13 在 Google Chrome 浏览器中的运行结果如图 2-13 所示。



图 2-13　applicationCount.jsp 运行结果

在 Google Chrome 浏览器中，页面刷新 3 次后，利用 Mozilla Firefox 浏览器查看到的结果如图 2-14 所示。

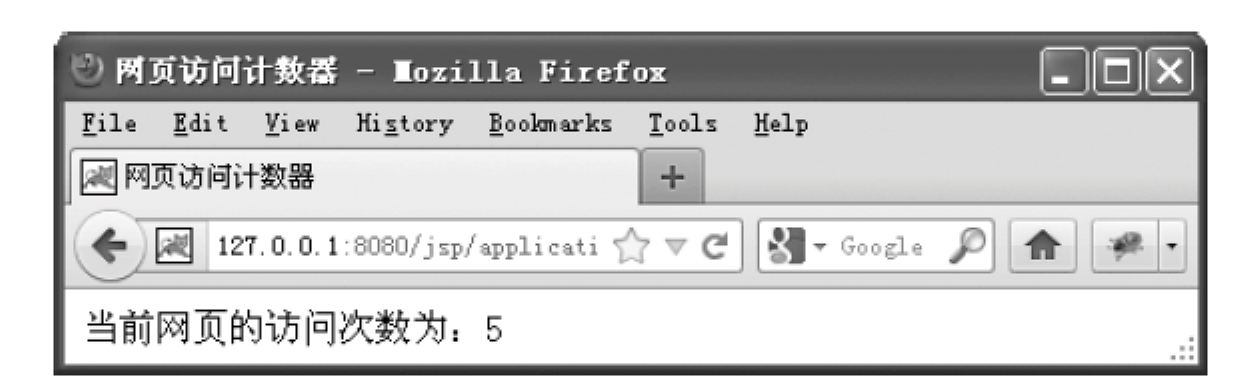


图 2-14　applicationCount.jsp 被访问第 5 次的结果

重新启动 Web 服务器后，再次浏览 applicationCount.jsp 的运行结果如图 2-15 所

示。尽管图 2-13 和图 2-15 中显示的网页访问次数都为 1，但二者访问的是不同的 application 对象中的属性。



图 2-15　重启 Web 服务器后，applicationCount.jsp 运行结果

2.6.6　cookie

cookie 是由 Web 服务器保存到客户端的一小段文本信息，可以随着用户请求和页面在 Web 服务器和浏览器之间传递。现在很多网站在用户浏览或者下载信息时都要求登录。有时登录了一次，再次访问该站点时，会自动识别出用户，这是因为上次登录的信息保存到了本地硬盘中，当用户再次访问该站点时，网页程序通过读取 cookie 识别出了用户。利用 cookie 可以为用户提供个性化服务，了解用户对当前网站的浏览习惯。

在 JSP 中实现跟踪用户的方法通常有四种：表单的隐藏域、URL 参数重写、cookie 和 session 会话。关于前两项方法，读者可以自行查阅相关文献。session 和 cookie 的区别在于：session 对象保存在 Web 服务器的内存中，可以保存普通类型的数据或者是一个 JavaBean 对象，当信息量比较大时会影响服务器的性能；cookie 存放在客户端的硬盘上，保存的数据只能是字符串形式，不同类型的 Web 浏览器会把 cookie 数据保存在硬盘的不同目录下。session 在会话开始时生成，会话结束或超时就会消失；cookie 可以长期保存在客户端，数据失效期取决于生成 cookie 时的设置。session 存放在服务器中，用户不能绕过 Web 程序进行修改；cookie 安全性较差，用户可以分析存放在本地的 cookie 并进行 cookie 欺骗。

严格地说，cookie 并不能算是 JSP 的内置对象。cookie 对象不能单独使用，必须与 request 对象或 response 对象结合使用。

cookie 对象的常用方法如表 2-6 所示。

表 2-6　cookie 对象的常用方法

方　法	说　明
getName()	返回 cookie 的名字
getValue()	返回 cookie 的值
setValue(String newValue)	cookie 创建后设置一个新的值
setMaxAge(int Age)	以秒计算，设置 cookie 的存在期限

修改代码 2-10 中的 login.jsp 文件源码，将表单的 action 属性的值改为 doLogin2.jsp。修改 doLogin.jsp，在用户登录时将登录用户名保存到 cookie 中，如代码 2-14 所示。

代码 2-14　增加了 cookie 信息的登录处理页面(doLogin2.jsp)

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%
    request.setCharacterEncoding("utf-8");
    String username=request.getParameter("username");
    String password=request.getParameter("password");
    if (username.equals("张三") && password.equals("123")){
        //生成一个 cookie,并保存登录的用户名
        Cookie cookie=new Cookie("username",username);
        //设置 cookie 的有效期为 60 秒
        cookie.setMaxAge(60);
        //将 cookie 添加到 response 对象中,以便写入客户端
        response.addCookie(cookie);
        response.sendRedirect("index2.jsp");
    }
    else
        response.sendRedirect("login.jsp");
%>
```

修改代码 2-12,将留言板首页修改成代码 2-15 的形式。新的留言板登录程序在用户访问 index2.jsp 时会读取 cookie,从中获得保存的用户名。如果 cookie 为空或者超过了有效期,会自动重定位到 login2.jsp。

代码 2-15　增加了读取 cookie 信息的留言板首页(index2.jsp)

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ page language="java" pageEncoding="UTF-8"%>
<%
    String username=null;
    //得到所有的 cookies
    Cookie[] cookies=request.getCookies();
    //如果设置了 cookie,得到它的值
    if(cookies!=null){
        for(int i=0;i<cookies.length;i++){
            if(cookies[i].getName().equals("username"))
                username=cookies[i].getValue();
        }
    }
    if(username==null)
        response.sendRedirect("login.jsp");
%>
<html>
    <head>
        <title>留言板程序</title>
    </head>
    <body>
        <h1>欢迎<font color="red"><%=username %></font>使用留言板程序</h1>
    </body>
</html>
```

同步训练

1. 编写一个建议的购物车程序，利用 session 保存用户选购的商品信息。提示：本程序可以包含两个页面：供顾客选购物品的 buy.jsp 页面和查看购物车的 result.jsp 页面。

2. 编写一个网页版聊天室程序。提示：可以通过设置 HTTP Header 实现页面定时刷新。例如，语句 response.setHeader("refresh","30") 将网页刷新间隔定为 30 秒。

3. 设计一个留言板注册页面，分别利用 Table 和 DIV 实现页面布局。页面参考样式如图 2-16 所示。



图 2-16　留言板注册页面样例

4. 编写注册信息接收页面，接收注册页面提交的所有信息，输出到浏览器中。

Chapter 3
第3章 深入 JSP

3.1 JavaBean

3.1.1 编写 JavaBean

JavaBean 是用 Java 编写的组件，每个 JavaBean 实现了一个特定的功能，其他开发者可以通过 JSP 页面、Servlet 来使用这些对象。JavaBean 的优点在于提高代码的重用性，可以一次性编写，任何地方重用。

JavaBean 通常具有如下特点。

(1) JavaBean 类必是一个 public 类。

(2) JavaBean 的属性必须为 private 类型。

(3) 对 JavaBean 属性的操作通常由两个 public 类型的方法 getXxx()和 setXxx()完成。其中，getXxx()方法用于获取属性 xxx 的值，setXxx()方法用于设置属性 xxx 的值。

因为 JavaBean 本质上就是一个 Java 类，因此可以在 JavaBean 中定义各种方法。但在实际的 JSP 程序设计时，通常只在 JavaBean 中包括构造函数和属性的 get/set 方法。

代码 3-1 声明了一个 JavaBean 类 note.java。

代码 3-1 实现 JavaBean

```
//note.java 文件源代码
package notes.model;
public class note {
    String title;
    String msgPerson;
    String content;

    public String getTitle() {
        try {
            byte b[]=title.getBytes("ISO-8859-1");
            title=new String(b);
            return title;
        } catch (Exception e) {
            return title;
        }
    }
```

```
        public void setTitle(String title) {
            this.title=title;
        }
        public String getMsgPerson() {
            try {
                byte b[]=msgPerson.getBytes("ISO-8859-1");
                msgPerson=new String(b);
                return msgPerson;
            } catch (Exception e) {
                return msgPerson;
            }
        }
        public void setMsgPerson(String msgPerson) {
            this.msgPerson=msgPerson;
        }
        public String getContent() {
            try {
                byte b[]=content.getBytes("ISO-8859-1");
                content=new String(b);
                return content;
            } catch (Exception e) {
                return content;
            }
        }
        public void setContent(String content) {
            this.content=content;
        }
    }
```

3.1.2　使用 JavaBean

要在 JSP 页面中使用编写好的 JavaBean，需要先在 JSP 页面中创建一个 JavaBean 的实例，然后才能通过该实例调用 JavaBean 定义的方法。使用 JavaBean 的方法有两种：一种是利用 JSP 定义的<jsp:useBean>动作、<jsp:setProperty>动作和<jsp:getProperty>动作；另一种是 new 操作显式地创建 JavaBean 实例。下面逐一介绍。

1. 标准的 JavaBean 操作动作

(1) <jsp:useBean>动作

<jsp:useBean>用于在 JSP 页面中创建一个 JavaBean 实例，语法格式如下：

```
<jsp:useBean id="bean名字" scope="page|request|session|application"
        class="类名">
```

其中：

① id 属性用于定义要加载的 JavaBean 实例的名称。使用该名称可以引用 JavaBean 的属性和方法。

② scope 属性定义 JavaBean 的生命周期，默认值为 page，表示该 JavaBean 只在当前

JSP 页面中可用。程序员可以根据需要将 scope 设置为其他值。

③ class 属性指定要加载的 JavaBean 的类文件路径。

(2) <jsp:setProperty>动作

<jsp:setProperty>动作和<jsp:useBean>动作配合使用,用于设置 JavaBean 的属性值,语法格式如下:

```
<jsp:setProperty name="beanname" property="*">
```

或

```
<jsp:setProperty name="beanname" property="属性名" [param="参数名"]>
```

或

```
<jsp:setProperty name="beanname" property="属性名" value="属性值">
```

其中:

① name 属性是通过<jsp:useBean>动作引入的 JavaBean 实例的名字。

② property 属性用于匹配 JavaBean 定义的属性。当取值为"*"时,JSP 容器会自动将 request 中各参数的值赋值给 JavaBean 实例中的同名属性;如果 request 中的属性名与 JavaBean 中的属性名字不相同,必须利用 param 指定 request 中的参数名。

③ 格式 3 利用 value 的值来设置 JavaBean 的属性值。

(3) <jsp:getProperty>动作

<jsp:getProperty>动作也必须和<jsp:useBean>动作配合使用,用于获取 JavaBean 中属性的值,所得到的值会自动转换成字符串类型,并通过输出流输出到 JSP 页面。格式如下:

```
<jsp:getProperty name="beanname" property="属性名">
```

各属性值的含义和<jsp:setProperty>动作中属性的含义相同。

代码 3-2 演示了如何利用标准的 JavaBean 操作动作访问代码 3-1 中定义的 JavaBean。

代码 3-2 使用 JavaBean

```
<!--post.jsp 文件源代码-->
<%@ page language="java" import="java.util.*" pageEncoding="GB2312"%>
<html>
    <head>
        <title>留言板程序</title>
    </head>
    <body>
        <form action="showPost.jsp" mrthod="post">
          标题:<input type="text" name="title" size="80"/><br/>
          留言人:<input type="text" name="msgPerson" size="80"><br/>
          内容:
        <textarea rows="20" cols="70" name="content"></textarea><br/>
```

```
            <input type="submit" value="留言"/>
            <input type="reset" value="重写"/>
        </form>
    </body>
</html>

<!-- showPost.jsp 文件源代码 -->
<%@ page language="java" contentType="text/html; charset=GB2312"%>
<html>
    <head>
        <title>显示留言</title>
    </head>
    <body>
    <jsp:useBean id="note" scope="session" class="notes.model.note"/>
    <jsp:setProperty property="*" name="note"/>
    <center><font color="red">您好,以下是您输入的留言信息:</font>
        <hr/>
        留言人:<jsp:getProperty property="msgPerson" name="note"/><br/>
        题目:<jsp:getProperty property="title" name="note"/><br/>
        内容:<jsp:getProperty property="content" name="note"/><br/>
    </center></body>
</html>
```

在 JSP 程序开发中,如果通过表单提交的数据中存在中文,则获取该数据后输出到页面时会显示乱码。因此,在输出获取的表单数据之前,必须进行转码操作。前面曾经利用 request.setCharacterEncoding("utf-8")进行过转码,但是当使用<jsp:setProperty>动作接收数据时,这种方法不会发挥作用,这时可以将该转码操作在 JavaBean 中实现,如代码 3-1 所示。

2 利用 new 操作实例化 JavaBean

JavaBean 类可以像普通类一样通过 import 语句引入 JSP 文件,再利用 new 操作显式创建实例。

在代码 3-3 中,给出了显式实例化 JavaBean 的方法。这里重写了 note.java 类,更名为 note2.java。note2.java 不需要使用 getBytes("ISO-8859-1")进行转码。

代码 3-3　显式实例化 JavaBean

```
//note2.java 文件源代码
package notes.model;
public class note2 {
    String title;
    String msgPerson;
    String content;
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title=title;
```

```
        }
        public String getMsgPerson() {
            return msgPerson;
        }
        public void setMsgPerson(String msgPerson) {
            this.msgPerson=msgPerson;
        }
        public String getContent() {
            return content;
        }
        public void setContent(String content) {
            this.content=content;
        }
}

<!-showPost2.jsp文件源代码-->
<%@page language="java" import="notes.model.note2" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>显示留言</title>
    </head>
    <body>
      <%
            request.setCharacterEncoding("UTF-8");
            note2 n=new note2();
            n.setTitle(request.getParameter("title"));
            n.setMsgPerson(request.getParameter("msgPerson"));
            n.setContent(request.getParameter("content"));

            String title=n.getTitle();
            String msgPerson=n.getMsgPerson();
            String content=n.getContent();
      %>
      <center><font color="red">您好,以下是您输入的留言信息:</font>
      <hr/>
            留言人:<%=msgPerson %><br/>
            题目:<%=title%><br/>
            内容:<%=content%><br/>
      </center>
    </body>
</html>
```

3.2 Servlet

3.2.1 Servlet 概念

Servlet 是运行在 Web 服务器端的 Java 小程序,可以生成动态的网页。在基于

MVC 的 Web 程序开发中，通常利用 Servlet 接受客户请求，调用业务逻辑处理的方法，最后通过 HttpResponse 对象跳转到相应的页面。

Servlet 与 JSP 的功能相似，都是用于处理客户请求并返回响应。事实上，对于每一个 JSP 页面，在第一次被客户访问时，服务器都会将其转换成 Servlet。例如，名为 notes 的 Web 工程的根目录下有个 post.jsp 页面（见代码 3-2），其对应的 Servlet 位于 Tomcat 主目录下的 work\Catalina\localhost\notes\org\apache\jsp 子目录中，名为 post_jsp.java，翻译后的内容如图 3-1 所示。

```
post_jsp.java - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletR
      throws java.io.IOException, javax.servlet.ServletException {

  final javax.servlet.jsp.PageContext pageContext;
  javax.servlet.http.HttpSession session = null;
  final javax.servlet.ServletContext application;
  final javax.servlet.ServletConfig config;
  javax.servlet.jsp.JspWriter out = null;
  final java.lang.Object page = this;
  javax.servlet.jsp.JspWriter _jspx_out = null;
  javax.servlet.jsp.PageContext _jspx_page_context = null;


  try {
    response.setContentType("text/html;charset-UTF-8");
    pageContext = _jspxFactory.getPageContext(this, request, response,
                  null, true, 8192, true);
    _jspx_page_context - pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;

    out.write("\r\n");
    out.write("<html>\r\n");
    out.write("  <head>\r\n");
    out.write("    <title>留言板程序</title>\r\n");
    out.write("  </head>\r\n");
    out.write("  <body>\r\n");
    out.write("    <form action=\"showPost2.jsp\" method=\"post\">\r\n");
    out.write("                    标  题:<input type-\"text\" name-\"title\" size-\"80\"/> <br/>\r\n")
    out.write("                    留言人:<input type=\"text\" name=\"msgPerson\" size=\"80\"><br/>\r\n");
    out.write("                    内  容:<textarea rows=\"20\" cols=\"70\" name=\"content\"></textarea
    out.write("    \t<input type-\"submit\" value-\"留言\"/>      \r\n");
    out.write("    \t<input type=\"reset\"  value=\"重写\"/>\r\n");
    out.write("    </form>\r\n");
    out.write("  </body>\r\n");
    out.write("</html>\r\n");
  } catch (java.lang.Throwable t) {
```

图 3-1　post.jsp 转换得到的 post_jsp.java 代码

Servlet 和 JSP 两者最大的区别就是：Servlet 以 Java 程序为主，输出 HTML 代码时需要使用 out.println 函数，也就是说，Java 中内嵌 HTML；而 JSP 以 HTML 布局为主，需要写 Java 代码时，在页面中直接插入 Java 代码，即 HTML 中内嵌 Java。

与 Servlet 相比，JSP 最大的优点是能够很方便地实现页面布局，显示复杂页面。

与 JSP 相比，Servlet 最大的优点在于后台处理，在业务处理和数据运算方面更稳健。另外，像动态生成验证码图像、显示邮箱的利用率等功能，通常都可以利用 Servlet 实现。

因此，在实际的项目开发时，可以利用 JSP 技术实现页面表现，利用 Servlet 技术完成逻辑处理，将二者有机地结合在一起使用。

3.2.2　Servlet 生命周期

Servlet 的生命周期主要由 4 个过程组成。

1. 装载和创建 Servlet 实例

Servlet 容器负责将 Servlet 类加载到 Java 虚拟机中并初始化。加载和实例化可以发生在 Web 服务程序启动时，也可以在客户端首次提出对 Servlet 服务请求时。

2. 初始化

当容器装载 Servlet 时，它会运行 Servlet 的 init()方法。在 Servlet 生命周期中，init()方法只被执行一次。为了提高系统性能，可以在 init()方法中缓存一些静态的数据或完成一些只需要执行一次的、耗时的操作，例如初始化数据库连接、获取配置信息等。在初始化不成功时，Servlet 实例将抛出 ServletException 异常或 UnavailableException 异常来通知 Web 容器。ServletException 异常用于指明一般的初始化失败，例如没有找到初始化参数；UnavailableException 异常用于通知容器该 Servlet 实例不可用。

3. 执行

初始化完毕，Servlet 就可以通过 service()等方法为客户提供服务请求。service()方法是 Servlet 的核心，能够获得与服务请求对象有关的信息，对请求进行处理，访问系统资源，然后将生成的响应封装在响应对象中，并回传给客户端。service()在执行的过程中可能会调用 doPost()、doGet()或程序员自定义的方法。在 service()方法执行期间，如果发生错误，Servlet 实例将抛出 ServletException 异常或 UnavailableException 异常。当发生 UnavailableException 异常时，客户将接收到 HTTP 404(请求的资源不可用)响应或者 HTTP 503(服务器暂时忙，不能处理请求)响应。

4. 服务结束

当容器停止且卸载 Servlet 时，将执行 destroy()方法。通常情况下，不需要覆盖 destroy()方法，但是当需要完成如关闭数据库连接等资源回收工作时，可以覆盖它。在 destroy()方法调用之后，该 Servlet 实例将会被容器释放并被 Java 的垃圾收集器回收。当再次需要请求该 Servlet 时，容器会创建一个新的实例。

3.2.3 Servlet 编程接口

Servlet 规范中规定，所有的 Servlet 类必须实现 javax. servlet. Servlet 接口。Servlet 规范提供了该接口的两个通用实现类：javax. servlet. GenericServlet 和 javax. servlet. http. HttpServlet。因此，程序员在编写 Servlet 时，既可以直接实现 Servlet 接口，也可以扩展 GenericServlet 或 HttpServlet。由于目前 Servlet 容器都是基于 HTTP 协议的，所以继承 HttpServlet 是最方便、最快捷的开发方式。

HttpServlet 类中包含了 init()、destroy()和 service()方法，同时增加了一些附加的方法，这些方法可以供 service()自动调用。其中最重要的是 doGet()方法和 doPost()方法。

1. doGet()方法

doGet()方法用于处理 HTTP 的 GET 请求。当客户通过 HTML 表单发出一个 HTTP GET 请求或者直接请求一个 URL 时，doGet()方法被自动调用。与 GET 请求相

关的参数添加到 URL 的后面,并与这个请求一起发送。当不会修改服务器端的数据时,应该使用 doGet()方法。

doGet()的语法如下:

```
protected void doGet(
    HttpServletRequest request,   //指定 HttpServletRequest 对象,包含客户端请求
    HttpServletResponse response  //指定 HttpServletResponse 对象,包含 Servlet 响应
)throws ServletException, IOException
```

2. doPost()方法

doPost()方法用于处理 HTTP 的 POST 请求。当客户通过 HTML 表单发出一个 HTTP POST 请求时,doPost()方法被自动调用。与 POST 请求相关的参数作为一个单独的 HTTP 请求从客户端发送到服务器。当需要修改服务器端的数据时,应该使用 doPost()方法。

doPost()的语法如下:

```
protected void doPost(
    HttpServletRequest request,   //指定 HttpServletRequest 对象,包含客户端请求
    HttpServletResponse response  //指定 HttpServletResponse 对象,包含 Servlet 响应
)throws ServletException, IOException
```

3.2.4 编写和部署 Servlet

下面通过一个简单的小例子说明如何编写和部署 Servlet。

1. 编写 Servlet

代码 3-4 给出的 servlet. LoginServlet 用于登录时的身份验证。

代码 3-4 LoginServlet. jsp

```
package servlet;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
        //由于本程序中 doGet()的功能和 doPost()的功能一致,将直接调用 doPost()
        doPost(request,response);
    }
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException, IOException {
        //设置 request 字符编码
        request.setCharacterEncoding("utf-8");
        String path= request.getContextPath();
        String basePath= request.getScheme()+"://";
```

```
            basePath+=request.getServerName();
            basePath+=":";
            basePath+=request.getServerPort();
            basePath+=path+"/";

            //获取客户请求参数
            String username=request.getParameter("username");
            String password=request.getParameter("password");
            if (username.equals("张三") && password.equals("123")){
                //将用户名保存在 session 中
                HttpSession session=request.getSession();
                session.setAttribute("username",username);
                //重定向到 index.jsp 页面,注意:由于 index.jsp 位于网站根目录中,为了防止
                //出现路径错误,最好使用绝对路径
                response.sendRedirect(basePath+"index.jsp");
            }
            else
                response.sendRedirect(basePath+"login.jsp");
        }
    }
```

为验证上述程序,将前面给出的 login. jsp 文件的 action 值修改为/servlet/LoginServlet,然后发布到 Web 服务器上进行查看。

基于 Web 的服务程序在与客户的交互过程中完成了特定的系统功能。为了实现交互过程,Servlet 中必须能够获取客户的请求参数,能够完成服务响应,能够读取和修改 session,能够实现数据流输出。在 JSP 文件中,可以通过内置的 request、response 和 session 等对象来获取这些信息。尽管在 Servlet 中不能直接使用 JSP 提供的内置对象,但是在 service()、doGet()和 doPost()等方法中封装了类型为 HttpServletRequest 的对象 request 和类型为 HttpServletResponse 的对象 response。利用 request 对象可以获得与客户请求相关的信息,利用 response 对象可以完成服务响应处理。如果要完成与 session 有关的处理,必须获得封装在 request 或 response 中的 HttpSession 的实例。

2 部署 Servlet

在编写完 Servlet 之后,必须在 web. xml 中对它进行正确的配置之后才能访问。代码 3-5 给出配置了 LoginServlet 的 web. xml。

代码 3-5 在 web. xml 中配置 Servlet

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet><!--部署 servlet.LoginServlet-->
        <description>Authenticate users</description>
```

```
        <display-name>loginServlet</display-name>
        <servlet-name>LoginServlet</servlet-name>
        <servlet-class>servlet.LoginServlet</servlet-class>
    </servlet>
    <!-为 Servlet 作 URL 映射 -->
    <servlet-mapping>
        <servlet-name>LoginServlet</servlet-name>
        <url-pattern>/servlet/LoginServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

其中，<url-pattern>/servlet/LoginServlet</url-pattern>标签给出了访问该Servlet所要用到的URL，也就是为什么要将login.jsp文件的action值修改为/servlet/LoginServlet的原因。

MyEclipse 等 IDE 集成开发环境能够自动生成 Servlet 的结构框架和部署过程，程序员在开发时可以直接采用，提高编程效率。

3.2.5　Servlet 过滤器

1. 什么是过滤器

Servlet 过滤器(Filter)作为一种特殊的 Servlet，位于客户端和 Web 应用程序之间。利用 Servlet 过滤器可以在客户请求到达 Servlet 和 JSP 文件之前，或在服务响应到达客户端前完成一些附加的操作。多个过滤器形成一个过滤器链，过滤器链中不同过滤器的先后顺序由部署文件 web.xml 中过滤器映射<filter-mapping>的顺序决定。图 3-2 演示了过滤器链的工作机制。

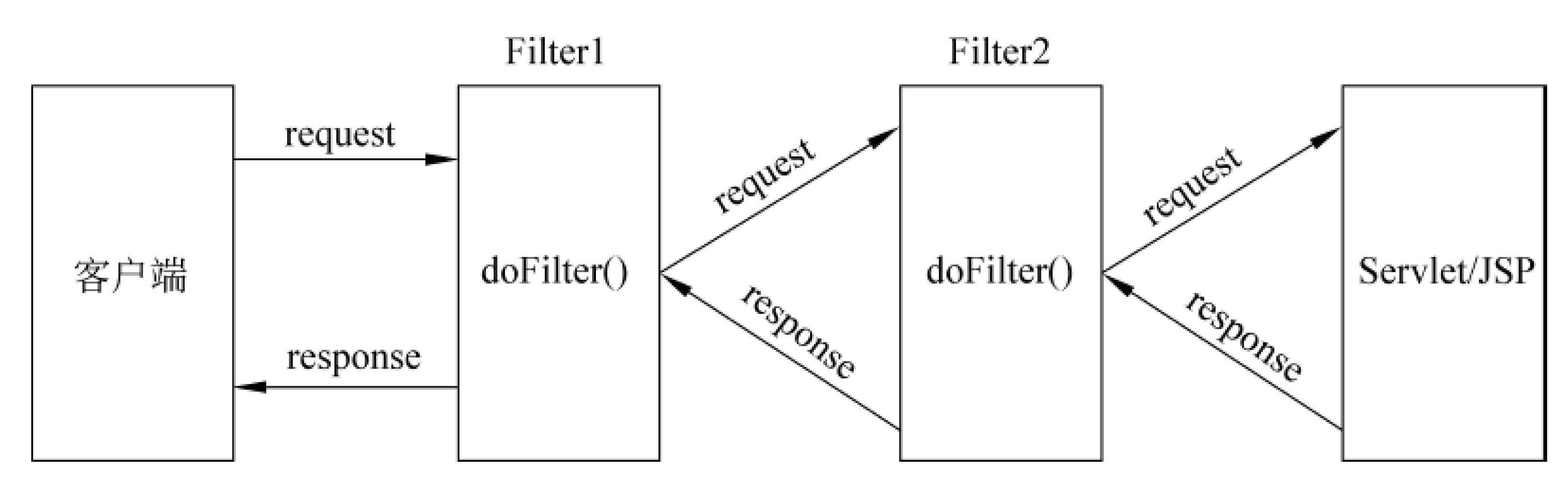


图 3-2　Servlet 过滤器链

利用 Servlet 过滤器可以完成以下工作。

(1) 字符编码转换：利用 Servlet 过滤器解决中文乱码问题。

(2) 权限验证：将 Web 服务程序的身份验证功能放到一个过滤器中，可以避免在每一个 Servlet 类和 JSP 文件中都要检查用户是否登录，身份是否合法等。

(3) 日志记录。

2 过滤器的编程接口

所有的 Servlet 过滤器类都必须实现 javax.servlet.Filter 接口。这个接口含有过滤器类必须实现的 3 个方法。

(1) public void init(FilterConfig config)。这是过滤器的初始化方法,Servlet 容器只有在实例化过滤器时才会调用该方法一次。该方法的参数是一个 FilterConfig 对象,包含与 Filter 相关的配置信息。

(2) public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)。每当请求和响应经过过滤器链时,容器都要调用一次该方法。当请求访问过滤器关联的 URL 时,Servlet 容器将先调用过滤器的 doFilter 方法,FilterChain 参数用于访问后续过滤器。

(3) public void destroy()。Servlet 容器通过调用 destroy()方法删除该过滤器。

Servlet 过滤器的创建步骤如下。

(1) 实现 javax.servlet.Filter 接口的 Servlet 类。

(2) 实现 init 方法,读取过滤器的初始化函数。

(3) 实现 doFilter 方法,完成对请求或过滤的响应。

(4) 调用 FilterChain 接口对象的 doFilter 方法,向后续的过滤器传递请求或响应。

(5) 在 web.xml 中配置 Filter。

3. 编写过滤器

客户端利用 HTTP 协议向服务器发送的数据请求,默认时采用的是 ISO-8859-1 格式编码,因此利用表单或者 URL 请求参数向 Servlet 和 JSP 文件传送数据时,其中的中文数据如果不加处理,在接收后显示时会出现乱码。为解决这一问题,可以编写一个过滤器将 Web 服务器接收到的客户请求转换成 UTF-8 格式。代码 3-6 给出了一个将客户请求转换成指定编码格式的过滤器。

代码 3-6 利用过滤器解决中文乱码(EncodeFilter.java)

```
package Filter;
import java.io.IOException;
import javax.servlet.* ;
public class EncodeFilter implements Filter {
    private FilterConfig config;
    private String encoding;
    public void doFilter(ServletRequest request,
            ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        if (encoding != null && !"".equals(encoding)) {
            //设置请求数据的编码方式
            request.setCharacterEncoding(encoding);
        }
        chain.doFilter(request, response);
    }
    public void destroy() {
        //TODO Auto- generated method stub
    }

    public void init(FilterConfig filterConfig) throws ServletException {
        //获取 Filter 的初始化参数的值 s
```

```
            this.config= filterConfig;
            this.encoding= filterConfig.getInitParameter("encoding");
        }
    }
```

4. 过滤器的配置

过滤器与普通的 Servlet 一样，也是在 web. xml 文件中配置的；不同的是，普通的 Servlet 要想发挥功能，需要客户端进行显式调用，而过滤器只需在 web. xml 中配置好就能够被容器自动加载。代码 3-7 给出 Filter. EncodeFilter 在 web. xml 中的配置示例。

代码 3-7　在 web. xml 中配置过滤器

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <filter>
        <filter-name>EncodeFilter</filter-name>
        <filter-class>Filter.EncodeFilter</filter-class>
            <init-param>
                <param-name>encoding</param-name>
                <param-value>UTF-8</param-value>
            </init-param>
    </filter>

    <filter-mapping><!--映射过滤器-->
        <filter-name>EncodeFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

其中，标签<filter-name>给出了过滤器的名字为 EncodeFilter；<filter-class>给出了过滤器 EncodeFilter 对应的 Java 类为 Filter. EncodeFilter. java；<init-param>定义了该过滤器包含一个初始化参数 encoding，值为 UTF-8，该参数由 Filter. EncodeFilter 实例在 init()方法中利用 FilterConfig 对象接收；<url-pattern>指出该过滤器适用于所有的客户请求。

3.3　JDBC

3.3.1　JDBC 工作原理

Web 应用程序离不开数据库的支持，利用数据库可以实现 Web 应用程序数据的持久化存储。基于 JSP 的应用程序在连接数据库时可以通过 JDBC(Java Data Base

Connectivity,Java数据库连接)技术来完成。JDBC提供了连接各种常用数据库的能力。JDBC向程序员提供了一组独立于数据库的API接口,这组接口提供了编写标准和考虑了不同应用程序设计的标准。接口的实现放在数据库驱动程序中。驱动程序负责将标准JDBC调用转变为特定数据库所需要的具体调用。每个驱动程序针对一种DBMS。图3-3给出了JDBC工作原理。

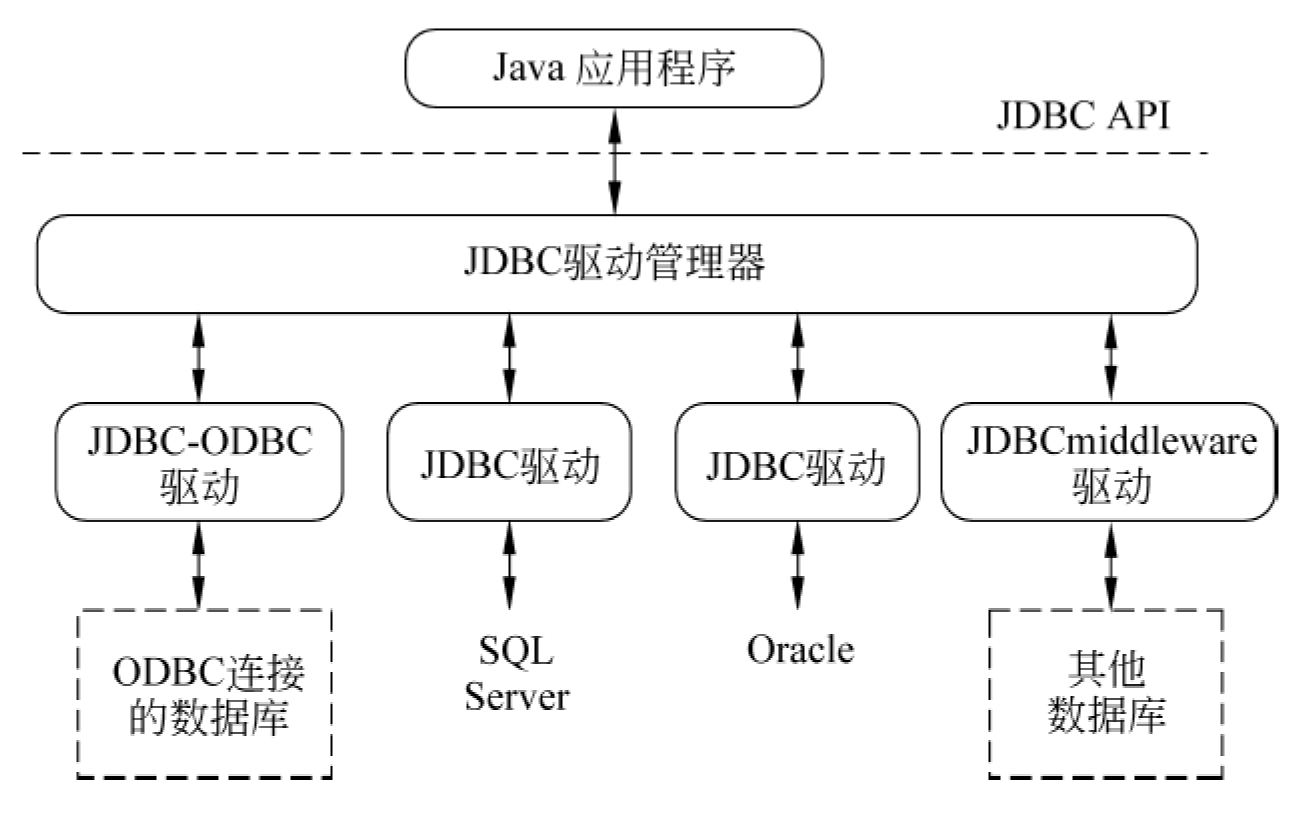


图3-3 JDBC工作原理

常见的JDBC连接方式分为以下4种类型。

(1) JDBC-ODBC桥:在客户端安装ODBC驱动程序,配置ODBC数据源,然后利用ODBC驱动程序提供JDBC访问。

(2) JDBC本地API:在客户端安装驱动程序,通过驱动程序将客户端API上的JDBC调用转换成对特定DBMS的调用。

(3) JDBC-middleware桥:该方式将JDBC访问操作直接转换成与DBMS无关的网络协议,通过该协议与某个特定的中间件服务器相联系,再由该服务器完成与DBMS之间的数据交换。该方法省去了在客户端安装驱动程序的麻烦,是最具灵活性的方式。

(4) 纯Java JDBC驱动:该方式利用DBMS提供的专用驱动程序把JDBC调用直接转换为对DBMS的操作。该方式在4种方法中的访问效率最高,也是目前最流行的方式。

在Web应用程序开发时,程序员只和上层的JDBC API打交道。通常,利用JDBC访问数据库的流程分为如下几个步骤。

(1) 加载JDBC驱动程序;

(2) 建立和数据库之间的连接;

(3) 创建表达式对象;

(4) 执行数据库的查询、添加、修改和删除等操作;

(5) 关闭数据库连接。

3.3.2 JDBC接口

JDBC定义了很多接口和类,主要包括DriverManager、Connection、Statement和ResultSet等。

1. 驱动程序管理器 DriverManager

java.sql.DriverManager 类用于管理 JDBC 驱动程序，负责跟踪可用的驱动程序，并在数据库和驱动程序之间建立连接。

DriverManager 类中最常用的方法为 getConnection(String url，String user，String password)。这是一个静态方法，用来获得数据库连接。方法的三个参数依次为要连接数据库的 URL、用户名和密码。该方法被调用时，DriverManager 类将定位指定的 Driver 类，并利用定位到的 Driver 类建立连接。连接成功时，将返回一个 java.sql.Connection 对象实例；如果不成功，将抛出 SQLException 异常。

2. 数据库连接接口 Connection

java.sql.Connection 接口负责与特定数据库的连接。通过该连接，程序员可以执行 SQL 语句并对返回的结果进行处理，也可以利用 getMetaData()方法获得数据库的元数据。Connection 接口的常用方法如表 3-1 所示。

表 3-1　Connection 接口的常用方法

方　法	说　明
createStatement()	创建并返回一个 Statement 实例，多用于执行不含有参数的 SQL 语句
prepareStatement()	创建并返回一个 PreparedStatement 实例，多用于执行包含参数的 SQL 语句
prepareCall()	创建并返回一个 CallableStatement 实例，利用该方法可以实现对数据库中定义的存储过程的调用
commit()	提交当前事务对数据库的所有更改，并释放 Connection 实例拥有的所有数据库加锁
rollback()	取消当前事务中的所有更改，并释放当前 Connection 实例拥有的所有数据库加锁。该方法仅适合非自动提交模式
close()	关闭数据库连接，并释放 Connection 实例占用的数据库和 JDBC 资源
isClosed()	查看当前的 Connection 实例是否被关闭
setAutoCommit()	设置当前 Connection 实例的提交模式，默认为 true，即自动将更改同步到数据库中。如果设为 false，需要通过执行 commit()或 rollback()方法手动将更改同步到数据库中
getAutoCommit()	查看当前的 Connection 实例是否处于自动提交模式
setSavepoint()	非自动提交模式下，在当前事务中创建并返回一个 Savepoint 实例
releaseSavepoint()	从当前事务中删除指定的 Savepoint 实例
setReadOnly()	设置当前 Connection 实例的读取模式，默认为非只读模式。不能在事务当中执行该操作，否则将抛出异常
isReadOnly()	查看当前的 Connection 实例是否为只读模式

3. 执行 SQL 语句接口 Statement

java.sql.Statement 接口用来执行静态的 SQL 语句，并返回执行结果。Statement 接口的常用方法如表 3-2 所示。

表 3-2 Statement 接口的常用方法

方　法	说　明
executeQuery(String sql)	执行 SELECT 语句,查询结果存储在一个 ResultSet 实例中
executeUpdate(String sql)	执行指 INSERT、UPDATE 或 DELETE 语句,返回结果为一个 int 型数值,用于说明数据库中受影响的记录的条数
addBatch(String sql)	将 INSERT 或 UPDATE 语句添加到 Batch 中。如果 JDBC 驱动程序不支持批量处理,将抛出异常
clearBatch()	清除 Batch 中的所有 SQL 语句,如果驱动程序不支持批量处理,将抛出异常
executeBatch()	执行 Batch 中的所有 SQL 语句,当全部执行成功时,返回一个更新计数数组,数组元素表示对应的 SQL 语句执行情况,其值可以为:①大于或等于零的数,表示 SQL 语句成功执行,为影响数据库中行数的更新计数;②－2,表示 SQL 语句成功执行,但没有得到受影响的行数;③－3,表示 SQL 语句执行失败。当驱动程序不支持批量,或者 Batch 中的某一个 SQL 语句未能成功执行时,将抛出异常
close()	关闭 Statement 实例,释放 Statement 实例占用的数据库和 JDBC 资源

4. 执行动态 SQL 语句接口 PreparedStatement

java.sql.PreparedStatement 接口是对 Statement 接口的扩展,用来执行包含参数的 SQL 语句。PreparedStatement 接口提供了一系列 setXxx 方法,用于对 SQL 语句中的参数赋值。

代码 3-8 演示了如何使用 PreparedStatement 接口执行动态的 SQL 语句。

代码 3-8 PreparedStatement 接口执行动态的 SQL 语句

```
String now=new java.util.Date();
java.text.SimpleDateFormat sdf=
new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
String sql="insert into notes(title,content,pubTime) values(?,?,?)";
PreparedStatement ps=conn.prepareStatement(sql);
ps.setString(1, "请教一个 JDBC 的问题");
ps.setString(2, "如何利用 PreparedStatement 执行动态 SQL?");
ps.setTimeStamp(3, sdf.parse(now));
int cnt=ps.executeUpdate();
```

需要注意的是,在对 SQL 语句中的参数赋值时,必须使用与输入参数的类型相兼容的 setXxx()方法。

5. 执行存储过程接口 CallableStatement

在进行 Web 开发时,通常会将一些常用的或很复杂的数据库操作定义为存储过程。利用 java.sql.CallableStatement 接口可以执行数据库中定义的存储过程。CallableStatement 是对 PreparedStatement 接口的扩展,可以通过 Connection 的 prepareCall()方法获取 CallableStatement 对象,形式如下:

```
con.prepareCall("{call 存储过程名(?,?)}");
```

其中，con 为 Connection 对象，问号为存储过程的参数，其值可以利用 setXxx()方法进行设置。对于具有输出参数的存储过程，需要利用 registerOutParameter()方法将结果参数注册为 OUT 类型，在存储过程执行后可以通过 getXxx 方法检索。

假设 SQL Server 数据库中有一个用于计算两个数乘积的存储过程 mathtutor。

```
CREATE PROCEDURE dbo.mathtutor
@m1 smallint,
    @m2 smallint,
    @result smallint OUTPUT
AS
SET @result=@m1 * @m2
```

代码 3-9 演示了如何利用 CallableStatement 调用 mathtutor。

代码 3-9　调用存储过程(mathtutor.jsp)

```
<%@page language="java" pageEncoding="UTF-8"%>
<%@page import="java.sql.*"%>
<html>
    <head>
        <title>测试访问数据库存储过程</title>
    </head>
    <body>
      <%
      String DRIVER="net.sourceforge.jtds.jdbc.Driver";
      String URL ="jdbc:jtds:sqlserver://127.0.0.1:1433/notes";
      String DBNAME="sa";
      String DBPASS="123";
      Class.forName(DRIVER);                    //加载 JDBC 驱动
      try {
          //连接指定的数据库
          Connection conn=DriverManager.getConnection(URL,DBNAME,DBPASS);
          CallableStatement proc=null;
          //准备调用 dbo.mathtutor 存储过程
          proc=conn.prepareCall("{ call dbo.mathtutor(?,?,?)}");
          proc.setInt(1,20);                    //为存储过程第 1 个参数传值 20
          proc.setInt(2,30);                    //为存储过程第 2 个参数传值 30
          //存储过程第 3 个参数注册为 OUT 类型
          proc.registerOutParameter(3,Types.INTEGER);
          proc.execute();                       //执行存储过程
          int result=proc.getInt(3);            //获得存储过程的返回值
          out.println("20*30="+result);         //输出存储过程的返回结果
          conn.close();                         //关闭数据库连接
      } catch (SQLException exception) {
              exception.printStackTrace();
      }
      %>
    </body>
</html>
```

6. 访问结果集接口 ResultSet

java.sql.ResultSet 接口用于访问检索结果集，其形式可以看做一个内存表或是数据库中的游标，表的列名和类型对应执行查询数据库的语句目标列的列名和类型。

ResultSet 接口提供了一组方法：first()、last()、previous()和 next()，用于移动指向数据行的指针。最初，指针指向首记录的前方，利用 next()方法可以将指针移动到下一条记录；当移动到尾记录末尾时，将返回 false。默认时，ResultSet 对象不可以更新，只能通过 next()方法遍历结果集。如果需要，可以生成可滚动和可更新的 ResultSet 对象。

ResultSet 接口提供了一组从当前行检索不同类型列值的 getXxx()方法，每个方法都有两个重载方法，可以通过列的索引编号或列的名称检索。

3.3.3 连接数据库

要想使用 JDBC 访问数据库，首先必须安装 JDBC 驱动程序，大多数 DBMS 都有自己专用的驱动程序。假设数据库的名称为 notes，下面介绍三种常用的数据库服务器配置驱动程序的方法。

1. 访问 MS SQL Server 数据库

MS SQL Server 版本比较多，SQL Server 2000 版本在利用 JDBC 访问时必须安装 SP4 补丁，比较麻烦。建议初学者安装 SQL Server 2005 或更高版本，并且下载和安装 JDBC 驱动。JSP 应用程序的驱动程序通常放到 Web 应用目录的/WEB-INF/lib 下，该驱动程序只能供本 Web 应用程序使用。如果要让 Web 服务器上的所有 Web 应用共享驱动程序，可以将其复制到 Tomcat 安装目录的/lib/中(以 Tomcat 服务器为例)。

(1) 微软专用驱动的连接字符串格式

最新版本的 SQL Server 驱动程序的名称为 sqljdbc4.jar。这里以 SQL Server 2005 Express 版为例，说明如何配置 JDBC 驱动程序名称和连接字符串。

```
String DRIVER="com.microsoft.sqlserver.jdbc.SQLServerDriver";
String URL="jdbc:sqlserver://127.0.0.1\\sqlexpress:1433;DataBaseName=notes";
String DBNAME="sa";
String DBPASS="monday";
Class.forName(DRIVER);
Connection conn=DriverManager.getConnection(URL,DBNAME,DBPASS);
```

(2) JTDS 驱动的连接字符串格式

JTDS 是为 SQL Server 和 Sybase 提供的第三方 JDBC 驱动，与 Microsoft 专用驱动不同，JTDS 在连接不同版本的 SQL Server 时，其连接字符串完全一致，并且效率更高。jtds.jar 可以从 http://jtds.sourceforge.net/免费下载。

```
String DRIVER="net.sourceforge.jtds.jdbc.Driver";
String URL="jdbc:jtds:sqlserver://127.0.0.1:1433/notes";
String DBNAME="sa";
String DBPASS="123";
Class.forName(DRIVER);
```

```
Connection conn=DriverManager.getConnection(URL,DBNAME,DBPASS);
```

2. 访问 Oracle 数据库

Oracle 作为关系数据库中的“大哥大”，在 Web 应用开发中一直扮演着重要角色。Oracle 的 JDBC 驱动的最新版本为 ojdbc14.jar。

```
String DRIVER="oracle.jdbc.driver.OracleDriver";
String URL="jdbc:oracle:thin:@127.0.0.1:1521:ORCL";
String DBNAME=" notes ";
String DBPASS="123";
Class.forName(DRIVER);
Connection conn=DriverManager.getConnection(URL,DBNAME,DBPASS);
```

3. 访问 MySQL 数据库

MySQL 是一种开放源代码的关系型数据库管理系统，由于其强大的功能和小巧的身材而深受 Java 程序员喜爱，下载地址为 http://www.mysql.com/。MySQL JDBC 驱动程序的名称为 mysql-connector-java-x.x.x-bin.jar，其中 x.x.x 为版本号，截至作者编写本书时，最新的版本号为 5.1.20。

```
String DRIVER="com.mysql.jdbc.Driver";
String URL="jdbc:mysql://127.0.0.1:3306 notes ";
String DBNAME="root";
String DBPASS="monday";
Class.forName(DRIVER);
Connection conn=DriverManager.getConnection(URL,DBNAME,DBPASS);
```

3.3.4　数据库连接池

1. 什么是数据库连接池

利用 JDBC 访问数据库中的数据，如果每一次访问都要重新建立数据库连接，将延长数据库访问时间，并严重影响系统性能。另外，如果对数据库的连接数量不加控制，可能会出现超过数据库处理能力的连接数量和处理请求。当频繁发生这种操作时，系统的性能将下降，甚至崩溃。

数据库连接池正是为解决上述问题而提出的。所谓数据库连接池，就是当系统启动时，一次性建立起一定数量的数据库连接，一旦应用程序需要连接数据库时，就从连接池中取出一个连接分配给应用程序；访问数据库操作完成后，分配的连接重新交还给连接池。通过这种方式，允许应用程序重复使用一个现有的数据库连接，避免了每次读写数据库时都要装载驱动、建立连接的过程，对于读写数据库操作比较频繁的系统，可以大大提高整体性能。

因此，使用连接池具有下列优点。

(1) 采用数据库连接池，避免了每次数据库访问时都要重新建立数据库连接，重新进行身份认证的过程，从而节省了时间，提高了系统性能。

(2) 解决了数据库对连接数量限制的问题。

（3）允许应用程序重复使用同一数据库连接。

当然，连接池也有一定的缺陷。由于服务器启动时就会创建一定数量的数据库连接，当请求数目远远少于连接数目时，将导致系统中存在很多空连接，它们会消耗一定的系统资源。在配置连接池时，可以通过设置 maxIdle 参数来减少由于空连接而造成的系统资源浪费。一旦系统中的空连接数量超过 maxIdle 时，系统将自动释放超过部分的连接。

2. Tomcat 连接池的配置与访问

现在主流的 Web 服务器都提供了数据库连接管理服务。下面以 Tomcat 服务器为例，介绍数据库连接池的设置和使用方法。

（1）正确安装 JDBC 驱动程序。将 JDBC 驱动程序复制到 Web 应用目录的/WEB-INF/lib 或 Tomcat 安装目录的/lib/中。

（2）配置连接池。在配置 Tomcat 连接池时使用了 Java 提供的 JNDI 技术。所谓 JNDI(Java Naming and Directory Interface，Java 命名和目录接口)，是一组在 Java 应用中访问命名和目录服务的 API。利用 JNDI，应用程序很容易根据连接池名称查找到连接池对象。

Tomcat 连接池的配置有两种方法。一种方法是将配置文件保存在 Web 应用目录的/META-INF/中；另一种方法是将配置信息放在 Tomcat 安装目录/conf/context. xml 文件中。

下面以 MySQL 数据库为例，介绍采用第一种方法配置连接池的过程。在 Web 应用目录的/META-INF/中建立 context. xml 文件，如代码 3-10 所示。

代码 3-10　连接池配置文件 context. xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
    <Resource
        name="jdbc/notes"
        auth="Container"
        type="javax.sql.DataSource"
        maxActive="100"
        maxIdle="30"
        maxWait="10000"
        username="root"
        password="123"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://127.0.0.1:3306/notes"
    />
</Context>
```

其中，每个字段的属性的含义如下。

① name：JNDI 名称，也是数据库连接池的名称，这里取值为 jdbc/notes。在应用程序中可以通过 jdbc/notes 申请数据库连接。

② auth：数据源的管理者，有两个可选值 Container 和 Application。Container 表示由容器来创建和管理数据源，Application 表示由 Web 应用来创建和管理数据源。

③ type：数据源的类型。

④ username：连接数据库的用户名。

⑤ password：连接数据库的密码。

⑥ maxActive：连接池中处于活动状态的数据库连接的最大数目。当取值为 0 时，表示不受限制。

⑦ maxIdle：连接池中处于空闲状态的数据库连接的最大数目。当取值为 0 时，表示不受限制。

⑧ maxWait：最长建立连接等待时间，单位为 ms。如果超过该时间，将发生异常；当取值为－1 时，表示无限制。

⑨ driverClassName：连接数据库的 JDBC 驱动程序。

⑩ url：连接数据库的路径。

除了上述方法外，还可以在 Tomcat 安装目录/conf/的 context.xml 文件中配置连接池。方法是将代码 3-10 中<Resource></Resource>标签及其下面的所有内容复制到 context.xml 的<context></context>标签下。

(3) 在配置完数据库连接池之后，就可以在 JSP、JavaBean 和 Servlet 中利用 JNDI 访问数据源了。代码 3-11 演示了在 JSP 中使用连接池的方法。

代码 3-11　测试数据库连接池(dspool.jsp)

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ page import="java.sql.* " %>
<%@ page import="javax.sql.DataSource" %>
<%@ page import="javax.naming.* " %>
<html>
    <head>
        <title>测试数据库连接池</title>
    </head>
    <body>
      <%
          Connection conn=null;
          try {
            Context ic=new InitialContext();
            //查找 JNDI,获得 context 中配置的连接池对象 jdbc/notes
            DataSource ds=(DataSource)ic.lookup("java:comp/env/jdbc/notes");
            //从连接池中获得数据库连接
            conn=ds.getConnection();
                    String sql="select userName,password from users";
            Statement stmt=conn.createStatement();    //创建 statement
            ResultSet rs=stmt.executeQuery(sql);      //执行 Select 查询
            while(rs.next()){
                out.println("用户名:"+rs.getString("userName"));
                out.println("   ");
```

```
                out.println("密码："+ rs.getString("password"));
                out.println("<br/>");
            }
            rs.close();
            stmt.close();
            conn.close();        //注意，此处是将连接返还给连接池
        } catch (SQLException exception) {
            exception.printStackTrace();
        } catch (NamingException namingException) {
            namingException.printStackTrace();
        }
    %>
  </body>
</html>
```

在代码 3-11 中，通过 JNDI 得到配置好的数据源 DataSource 对象 ds，然后利用 ds.getConnection()从连接池中申请连接，最后利用 conn.close()将连接返还给连接池。

3.4 JSP MVC 编程

3.4.1 MVC 设计思想

MVC(Model-View-Controller)是模型(Model)、视图(View)和控制(Controller)的缩写，是在系统开发中最为流行的设计模式之一。利用 MVC 模式，可以实现 Web 应用程序的职能分工。图 3-4 给出了 MVC 模式中各个部件之间的关系。

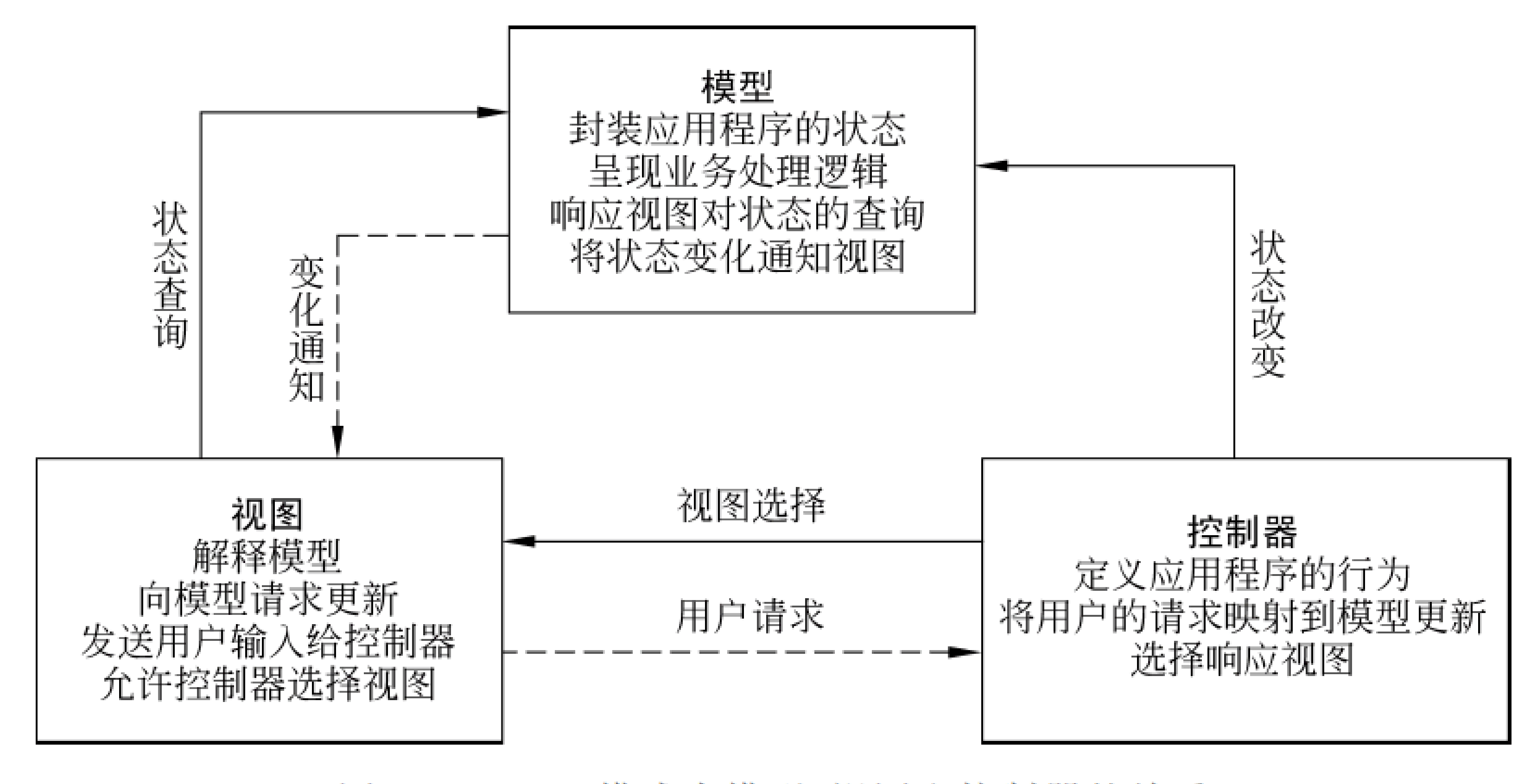


图 3-4　MVC 模式中模型、视图和控制器的关系

模型是 MVC 模式的核心，它封装了系统的核心数据、业务处理逻辑和功能的计算关系，独立于具体的界面表达和 I/O 操作。例如，Excel 中的一组数据既可以显示为表格，也可以显示为饼图或柱状图等。正是由于被模型封装的数据独立于其显示方式，所以应用于模型的代码只需写一次就可以被多个视图重用，减少了代码的重复性。

视图代表用户交互界面。对于 Web 应用程序来说，视图就是由 HTML、Adobe Flash、XHTML、XML/XSL 和 WML 等元素构成的界面。一个规模比较大的应用系统，其界面的处理也变得具有挑战性。同一个数据模型可能有很多不同的视图，利用 MVC 设计模式对于视图的处理仅限于视图上数据的采集、处理和接收用户的请求，不包括对这些数据和请求的业务处理。业务的处理是由模型完成的。例如，在基于 MVC 架构实现的留言板程序中，显示留言的视图只接受来自模型的数据并显示给用户，而创建留言的视图只负责将用户界面的输入数据和请求传递给控制和模型。

控制器定义了视图对用户输入的响应方式，它不负责任何数据处理。当控制器接收到一个用户请求时，它只是负责决定调用哪个模型去处理该请求。因此，控制器是协调模型和视图工作的部件。

MVC 模式使得模型和视图相互独立，可以把一个模型独立地移植到新的平台工作，需要做的只是在新平台上对视图和控制器进行新的修改。MVC 模式使得模型中的数据发生变化时，所有依赖于该数据的视图都会反映出这些变化，从而使所有关联的视图和控制器做到行为同步。利用 MVC 模式，可以将应用程序分割成若干逻辑部件，每个部件交由不同的开发人员去完成，这有助于实现团队协作开发。

在 JSP 开发过程中，基于 MVC 模式的 Web 开发框架包括 Struts 2、Spring、JSF 和 Tapestry 等。

3.4.2　MVC 模式实现

在早期的 JSP 应用程序开发中，模型的角色通常由 JSP 页面或 HTML 页面来充当，视图的角色由 JavaBean 来充当，控制器的角色可以利用 Servlet 来实现。图 3-5 给出了基于 Servlet 的 MVC 模式的架构。

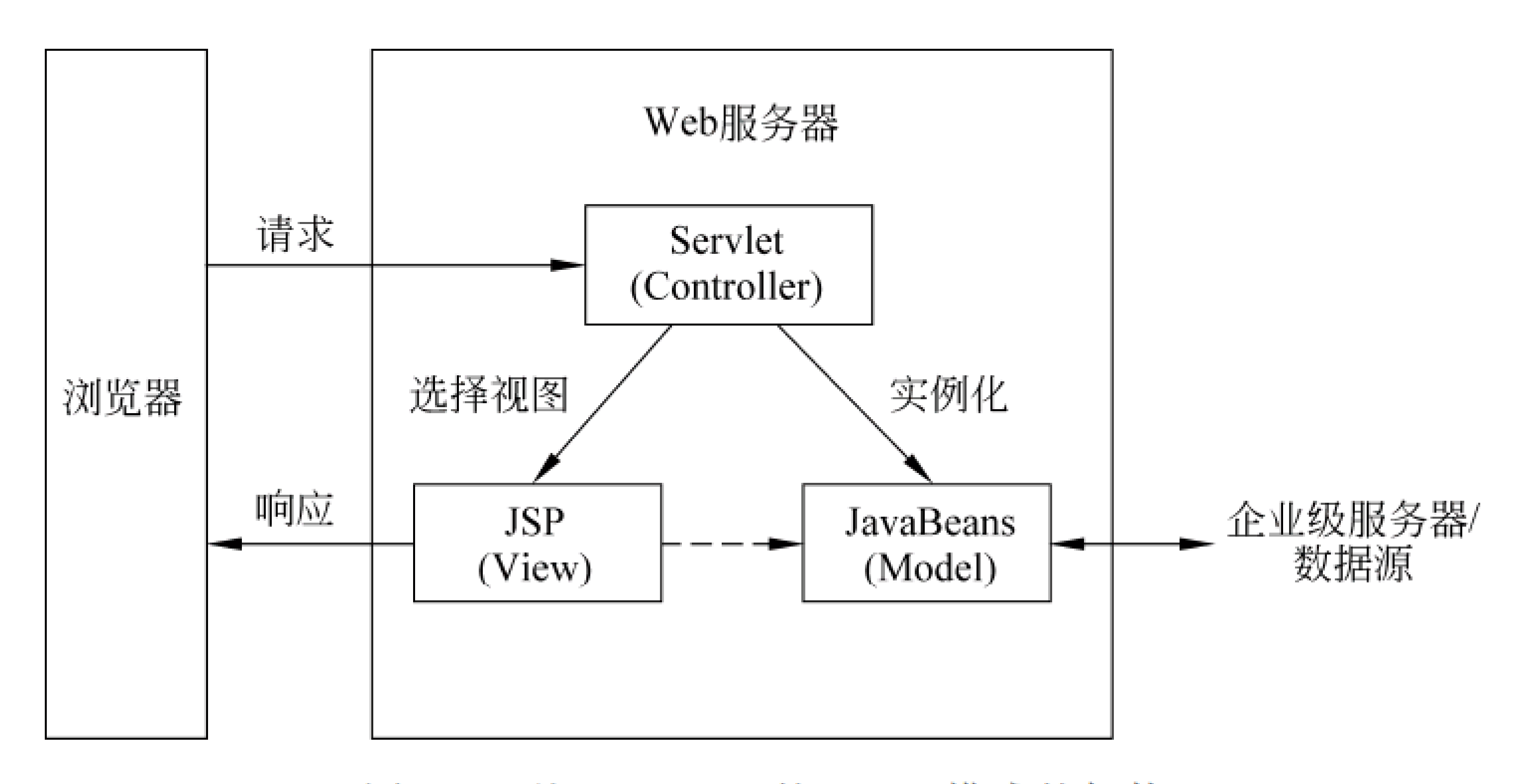


图 3-5　基于 Servlet 的 MVC 模式的架构

Servlet 负责接收用户的请求，并将请求分发给响应的视图来产生响应。Servlet 控制器还根据 JSP 视图的需求生成 JavaBean 的实例并输出给 JSP 页面。JSP 视图可以通过直接调用 JavaBean 实例的方法或使用＜jsp:getProperty＞动作获得 JavaBean 中的数据。

接下来通过一个实例演示如何利用 JSP＋Servlet＋JavaBean 实现 MVC 模式。为了

方便读者理解，图 3-6 给出了本实例在 myEclipse 集成开发环境中的目录结构。请大家认真查看 JSP 文件和 Servlet 文件所在的目录，体会为什么 LoginServlet 类重定向页面的 URL 路径要写成/success.jsp 和/failure.jsp 的形式，而 failure.jsp 页面中超级链接的 URL 路径要写成/Login/login.jsp 的形式。能否采取其他形式书写各个页面的路径呢？

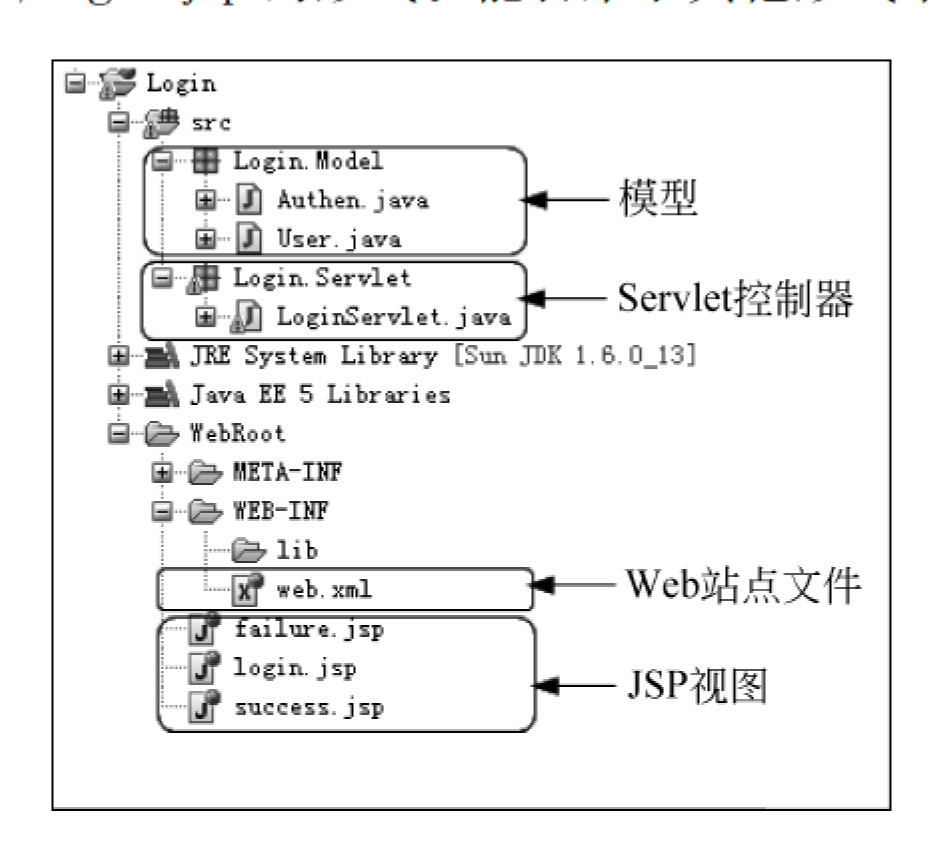


图 3-6　基于 Servlet 的 MVC 实例的目录结构

该例子中包括两个 Bean 类：User 类和 Authen 类。User 对象负责保存登录用户的信息，Authen 对象负责实现对登录用户的信息进行验证的业务逻辑。代码 3-12 给出了 User 类和 Authen 类的代码片段。

代码 3-12　模型 User.java 和 Authen.java

```
/* User.java */
package Login.Model;

public class User {
    private String username;              //用户名
    private String password;              //密码
    …
    //省略了 username 和 password 的 getter 和 setter 方法
}

/* Authen.java */
package Login.Model;
public class Authen {
    private User user;
    public Authen(User user){
        this.user=user;
    }

    …
    //省略了 user 的 getter 和 setter 方法
    public boolean validate(){
        String username=user.getUsername();
```

```
        String password=user.getPassword();
        //简化了业务处理逻辑,直接判断了用户名和密码
        //真正的系统中,需要查询数据库进行验证
        if("zhangsan".equals(username) &&"123".equals(password))
            return true;
        else
            return false;
    }
}
```

视图部分由三个 JSP 页面组成：login.jsp、success.jsp 和 failure.jsp。login.jsp 负责显示一个登录表单；Success.jsp 负责在登录成功后向用户显示欢迎信息，这里直接利用<jsp:userBean>动作取出了保存在 session 中的 user 对象；failure.jsp 在登录信息未通过验证后显示错误提示。代码 3-13 给出了视图部分的代码。

代码 3-13　视图页面 login.jsp、success.jsp 和 failure.jsp

```
<!--login.jsp-->
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>登录</title>
    </head>
    <body>
        <h2>请您登录</h2>
        <form name="form1" action="servlet/LoginServlet" method="post">
            用户名:<input type="text" name="username"><br>
            口令:<input type="password" name="password"><br>
            <input type="submit" value="提交">
            <input type="reset" value="重置">
        </form>
    </body>
</html>

<!--success.jsp-->
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ page import="Login.Model.User" %>
<html>
    <head>
        <title>登录成功</title>
    </head>
    <body>
        <jsp:useBean id="user" class="Login.Model.User" scope="session"/>
        <h2>
        <jsp:getProperty property="username" name="user"/>您好,欢迎登录系统!
        </h2>
    </body>
```

```
</html>

<!-- failure.jsp-->
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>登录失败</title>
    </head>
    <body>
        <h2>用户名或者口令不正确,请
            <a href="/Login/login.jsp">重新登录!</a></h2>
    </body>
</html>
```

LoginServlet 类充当了控制器,它接收 Login.jsp 页面发送来的用户请求,实例化 User 对象保存用户登录信息,实例化 Authen 类对用户登录信息进行验证,然后根据验证的结果选择对应的视图响应给用户。代码 3-14 给出了 LoginServlet 类的代码。

代码 3-14　视图页面 login.jsp、success.jsp 和 failure.jsp

```
package Login.Servlet;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.*;
import Login.Model.*;
public class LoginServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
        doPost(request,response);
    }

    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                       throws ServletException, IOException {
        String username=request.getParameter("username");
        String password=request.getParameter("password");
        User user=new User();                    //创建模型对象
        user.setUsername(username);

        Authen authen=new Authen(user);
        String forward;                          //保存要转向的视图文件
        //如果登录成功,把用户名写入 session,并且转向 success.jsp,
        //否则转向 failure.jsp
        if(authen.validate()){                   //调用 Authen 对象的方法进行验证
            HttpSession session=(HttpSession)request.getSession(true);
```

```
            session.setAttribute("user",user);        //把用户保存到 session 中
            forward="/success.jsp";                   //指定目标转向文件为 success.jsp
        }
        else
            forward="/failure.jsp";                   //指定目标转向文件为 failure.jsp
        RequestDispatcher dispatcher=request.getRequestDispatcher(forward);
        dispatcher.forward(request,response);         //完成跳转
    }
}
```

代码 3-15 给出了 web.xml 中 LoginServlet 的配置。

代码 3-15　web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <description>Authenticate users</description>
        <display-name>loginServlet</display-name>
        <servlet-name>LoginServlet</servlet-name>
        <servlet-class>Login.Servlet.LoginServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>LoginServlet</servlet-name>
        <url-pattern>/servlet/LoginServlet</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>login.jsp</welcome-file><!--欢迎页面-->
    </welcome-file-list>
</web-app>
```

3.5　JSP 的错误处理

JSP 中的错误有两种：一种是编译错误，当 Web 容器在编译由 JSP 页面转换成 Servlet 源文件时，如果 Servlet 文件无法通过 Java 编译检查时，将会发生这类错误，错误代码编号为 500；另一种是运行时出现的错误，当 Web 容器在执行已编译的 Java 字节码来处理一个到来的请求时出现。

尽管 Web 容器对于 JSP 中的错误会给出默认的处理，但有时制作一个自己的错误处理页面会显得更人性化。

1. 利用 page 指令捕捉编译错误

JSP 编译错误的捕捉可以通过错误页面转发来完成。采用这种方法，首先要制作一个错误处理页面 error.jsp，并在页面的 page 指令中将 isErrorPage 属性设置为 true，然后在所有需要捕捉错误的页面中设置 page 指令。

```
<%@ page errorPage="error.jsp" %>
```

代码 3-16 给出了一个具有语法错误的页面 doLogin.jsp。注意，其中的"String password=request.getParameter("password")"语句后面缺少了一个分号";"，该页面将错误转发到 error.jsp 中进行处理。

代码 3-16　page 指令进行出错处理(doLogin.jsp)

```
<%@ page language="java" pageEncoding="UTF-8" errorPage="error.jsp"%>
<%
    String userName=request.getParameter("userName");
    String password=request.getParameter("password")

    if("zhangsan".equals(userName)&&"123".equals(password))
        response.sendRedirect("index.jsp");
%>
```

代码 3-17 给出了错误处理页面 error.jsp。注意，page 指令中的 isErrorPage 的值为 true。

代码 3-17　出错处理页面(error.jsp)

```
<%@ page language="java" pageEncoding="UTF-8" isErrorPage="true"%>
<html>
    <head>
        <title>留言板出错啦</title>
    </head>
    <body>
        <img alt="" src="images/logo.png">
        <h2>留言板程序出错啦!请与管理员联系</h2>
        <hr/>
        <%=exception.getMessage()%>
    </body>
</html>
```

如果运行 doLogin.jsp 程序，将出现如图 3-7 所示的界面。

2. 利用 Web.xml 配置错误处理页面

对于运行时发生的错误，无法利用 page 指令进行处理。常见的运行时错误包括 HTTP 400(请求无效)、HTTP 403(禁止访问)和 HTTP 404(无法找到文件)。程序员可以在站点文件 web.xml 中配置处理这一类错误的页面。

代码 3-18 给出了一个用于处理错误的 web.xml 实例。这里将 400 和 404 错误处理页面指定为 error.html。



图 3-7　出错处理页面运行结果

代码 3-18　利用 web.xml 配置错误处理

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
        <error-page>
            <error-code>500</error-code>
            <location>/error.jsp</location>
        </error-page>
        <error-page>
            <error-code>400</error-code>
            <location>/error.html</location>
        </error-page>
        <error-page>
            <error-code>404</error-code>
            <location>/error.html</location>
        </error-page>
</web-app>
```

值得一提的是，这种方法同样适用于处理编译时发生的错误。例如在代码 3-18 中，将 500 错误的处理页面设定为 error.jsp，此时就不必在所有的页面中再利用 page 指令的 errorPage 属性设置出错页面了。

3.6　案例 1：用 JSP 编写留言板程序

3.6.1　功能分析

本节将利用前面讲过的知识，设计一个简单的公共留言板程序。用户登录之后，可

以看到所有的留言标题的列表;点击某个留言标题,可以查看留言的内容;点击留言按钮,可以实现留言功能。

之前使用JDBC操作数据库,都是直接在JSP页面中写JDBC代码。这样导致JSP页面中包含大量的HTML代码和Java代码,显示和功能代码混在一起,维护比较困难。为解决该问题,将采用在实际开发中广泛使用的DAO设计模式来编写这个小程序。

DAO抽象与封装所有对数据源的访问;负责管理对数据源的连接,以及数据的存取。DAO包括五个重要的部分,分别如下。

(1) 数据库连接类:主要功能是连接数据库并获得连接对象,以及关闭数据库。通过数据库连接类可以大大地简化开发,在需要进行数据库连接时,只需创建该类的实例,并调用其中的方法就可以获得数据库连接对象和关闭数据库,不必重复操作。

(2) PO类:持久化类,包含的属性和数据库表中的字段一一对应,数据库中有多少个表就应该有多少个PO类。PO类是一个标准的JavaBean,每个属性都应提供get和set方法。

(3) DAO接口:定义了所有的用户操作,例如添加、删除和查询记录等。

(4) DAO实现类:实现了DAO接口。一个DAO实现类对应一个表,实现了与该表有关的所有操作。

(5) DAO工厂类:在没有DAO工厂类的情况下,必须通过创建DAO实现类的实例才能完成数据库操作,对于后期的修改非常不便,极端情况下可能会需要修改每一个DAO实现类的代码。使用DAO工厂类可以很好地解决后期修改的问题,可以通过该DAO工厂类的一个静态方法来获得DAO实现类实例。这时如果需要替换DAO实现类,只需修改该DAO工厂类中的方法代码,而不必修改所有的操作数据库代码。

限于读者目前掌握的知识,本程序在实现页面部分时仍然采用嵌入JSP脚本的形式。在学习Struts 2知识的过程中,我们将逐步将其改造,使之完全遵循MVC模式。

3.6.2 数据库结构

本程序将采用SQL Server 2005 Express作为数据库平台,共包括两张表格:留言表notes和用户表users,具体结构如表3-3和表3-4所示。

表3-3 notes表

属性名	类型	说明
noteId	int	留言ID,主码,默认IDENTITY(1,1)
title	varchar(32)	留言的主题
content	varchar(200)	留言的内容
pubTime	datetime	留言的时间
userId	int	留言人ID

表 3-4 users 表

属性名	类 型	说 明
userId	int	用户 ID,主码,默认 IDENTITY(1,1)
userName	varchar(20)	用户名
password	varchar(10)	密码
head	varchar(20)	头像
regTime	datetime	注册时间
gender	smallint	性别

3.6.3 实现 PO 类

每个数据表都应映射到一个持久化类。代码 3-19 给出了 notes 表和 users 表对应的 PO 类。

代码 3-19 PO 类

```
/* Notes.java */
package jNotes.bean;
import java.sql.Timestamp;
public class Notes {
    private int noteId;
    private String title;
    private String content;
    private Timestamp pubTime;
    private User user;
    /* 省略了 set 和 get 方法 */
}

/* Users.java */
package jNotes.bean;
import java.util.Date;
public class User implements{
    private Integer userId;
    private String userName;
    private String password;
    private String head;
    private Date regTime;
    private String email;
    private int gender;
    /* 省略了 set 和 get 方法 */
}
```

3.6.4 DAO 接口设计

DAO 接口层使得低级别的数据访问逻辑与高级别的业务逻辑分离,可以隐藏持久化操作的细节。代码 3-20 定义留言板程序的 DAO 类的接口,并在接口中定义所有的业

务方法和数据操作方法。

代码 3-20 留言板程序的 DAO 接口

```
/* NotesDao.java */
package jNotes.dao;
import jNotes.bean.Notes;
import java.util.List;
public interface NotesDao {
    public int addNote(Notes note);
    public List<Notes> getAllNotes();
    public Notes getNoteById(int noteId);
}

/* UserDao.java */
package jNotes.dao;
import jNotes.bean.User;
public interface UserDao {
    public User findUser(int userId);
    public User findUser(String userName,String password);
    public int addUser(User user);
}
```

NotesDao.java 定义了访问留言表的三个方法：用于添加新留言的 addNote()方法、用于获取所有留言列表的 getAllNotes()方法和用于根据留言 ID 获取留言详细信息的 getNotesById()方法。

UserDao.java 定义了访问用户表的三个方法：根据用户 ID 查找用户 findUser()方法、根据用户名和密码查找用户的 findUser()方法以及添加新用户的方法 addUser()。

3.6.5 数据库连接和 DAO 实现类

DAO 实现类用于实现 DAO 接口。为简化操作，本程序先定义了一个 BaseDao 类，提供了获取数据库连接操作 getConn()方法、关闭连接操作 closeAll()方法和执行数据库更新语句的 executeSQL()方法。代码 3-21 给出了 BaseDao 类的实现细节。

代码 3-21 BaseDao. Java

```
package jNotes.dao.impl;

import java.sql.*;
import javax.naming.*;
import javax.sql.DataSource;
public class BaseDao {
    public Connection getConn()
        throws ClassNotFoundException, SQLException{
        Connection conn=null;
            try{                                //根据 JNDI 获取数据源
            Context ic=new InitialContext();
            DataSource source=(DataSource)
```

```
            ic.lookup("java:comp/env/jdbc/notes");   //JNDI 名称 notes
        conn= source.getConnection();
    } catch (SQLException exception) {
        exception.printStackTrace();
    } catch (NamingException namingException) {
            namingException.printStackTrace();
    }
    return conn;
}

public void closeAll(Connection conn, PreparedStatement pstmt,
                     ResultSet rs){
    if(rs != null){                              //关闭结果集
        try { rs.close();}
        catch (SQLException e) {e.printStackTrace();}
    }
    if(pstmt != null){                           //关闭语句
        try { pstmt.close();}
        catch (SQLException e) {e.printStackTrace();}
    }
    if(conn != null){                            //关闭连接
        try { conn.close();}
        catch (SQLException e) {e.printStackTrace();}
    }
}

public int executeSQL(String preparedSql,String[] param) {
    Connection conn= null;
    PreparedStatement pstmt= null;
    int num = 0;
    try {                                        //构造 prepareStatement 语句的参数
        conn= getConn();
        pstmt= conn.prepareStatement(preparedSql);
        if(param != null) {
            for(int i= 0; i < param.length; i++) {
                pstmt.setString(i+1, param[i]);
            }
        }
        num= pstmt.executeUpdate();
    } catch (ClassNotFoundException e) {
    e.printStackTrace();
    } catch (SQLException e) {e.printStackTrace();}
    finally {closeAll(conn,pstmt,null);}
    return num;
}
}
```

NotesDaoImpl 类和 UserDaoImpl 类扩展该类，参见代码 3-22。

代码 3-22　DAO 实现类

```
/* NotesDaoImpl.java */
package jNotes.dao.impl;
import jNotes.bean.*;
import jNotes.dao.NotesDao;
import java.sql.*;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.Date;

public class NotesDaoImpl extends BaseDao implements NotesDao {
    private Connection conn=null;
    private PreparedStatement pstmt=null;
    private ResultSet rs=null;

    public int addNote(Notes note) {
        String sql="insert into notes(content, pubTime, title, userId) values(?,?,?,?) ";
        String time=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
                    .format(new Date());      //用服务器当前作为留言时间
        String[] param={note.getContent(),time,note.getTitle(),
                    note.getUser().getUserId().toString()
        };                                    //准备 SQL 语句的参数
        return this.executeSQL(sql, param);   //调用 BaseDao 的 executeSQL 方法
    }

    public List<Notes> getAllNotes() {
        List<Notes> list=new ArrayList<Notes>();
        String sql="select * from notes order by pubTime desc";
        try{
            conn=this.getConn();                       //获取数据库连接
            pstmt=conn.prepareStatement(sql);  //获取 PrepareStatement 对象
            rs=pstmt.executeQuery();                   //执行查询
            while(rs.next()){                 //依次取出查询结果中的每一条数据,保存到 list 中
                Notes note=new Notes();
                note.setNoteId(rs.getInt("noteId"));
                note.setTitle(rs.getString("title"));
                //获取当前留言人的信息
                User user=new UserDaoImpl().findUser(rs.getInt("userId"));
                note.setUser(user);
                list.add(note);                        //将一条留言保存到 list 中
            }
        }catch(Exception e){e.printStackTrace();}
        finally{this.closeAll(conn, pstmt, rs); }
        return list;
    }

    public Notes getNoteById(int noteId) {
```

```
            Notes note=new Notes();
            String sql="select * from notes where noteId="+noteId;
            try{
                conn=this.getConn();                //获取数据库连接
                pstmt=conn.prepareStatement(sql);   //获取 PrepareStatement 对象
                rs=pstmt.executeQuery();            //执行查询
                if(rs.next()){                          //结果不为空时,利用查询结果构造 Note 对象
                    note.setNoteId(rs.getInt("noteId"));
                    note.setContent(rs.getString("content"));
                    note.setPubTime(rs.getTimestamp("pubTime"));
                    note.setTitle(rs.getString("title"));
                    //获取当前留言人的信息
                    User user=new UserDaoImpl().findUser(rs.getInt("userId"));
                    note.setUser(user);
                }
            }catch(Exception e){ e.printStackTrace();}
            finally {this.closeAll(conn, pstmt, rs); }
            return note;
        }
}

/* UserDaoImpl.java */
package jNotes.dao.impl;
import jNotes.bean.User;
import jNotes.dao.UserDao;
import java.sql.*;
import java.text.SimpleDateFormat;
import java.util.Date;
public class UserDaoImpl extends BaseDao implements UserDao {
    private Connection conn=null;
    private PreparedStatement pstmt=null;
    private ResultSet rs=null;

    public int addUser(User user) {
        String sql="insert into users(userName, password, gender,
                head,regTime) values(?,?,"+user.getGender()+",?,?)";
        String time=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").
                format(new Date());             //用服务器当前时间作为用户注册时间
        String[] parm={user.getUserName(), user.getPassword(),
                user.getHead(),time };          //准备 SQL 语句的参数
        return this.executeSQL(sql, parm);      //调用 BaseDao 的 executeSQL 方法
    }

    public User findUser(int userId) {
        String sql="select * from users where userId=?";
        User user=null;
        try {
            conn=this.getConn();                //获取数据库连接
            pstmt=conn.prepareStatement(sql);   //获取 PrepareStatement 对象
```

```
            pstmt.setInt(1, userId);
            rs=pstmt.executeQuery();          //执行查询
            while(rs.next()) {                    //结果不为空时,利用查询结果构造 User 对象
                user=new User();
                user.setUserId(rs.getInt("userId"));
                user.setUserName(rs.getString("userName"));
                user.setPassword(rs.getString("password"));
                user.setGender(rs.getInt("gender"));
                user.setHead(rs.getString("head"));
                user.setRegTime(rs.getDate("regTime"));
            }
        } catch (Exception e) { e.printStackTrace();}
        finally {this.closeAll(conn, pstmt, rs); }
        return user;
    }

    public User findUser(String userName, String password) {
        String sql="select * from users where userName=? and password=?";
        User user=null;
        try {
            conn=this.getConn();              //获取数据库连接
            pstmt=conn.prepareStatement(sql); //获取 PrepareStatement 对象
            pstmt.setString(1, userName);
            pstmt.setString(2, password);
            rs=pstmt.executeQuery();          //执行查询
            while(rs.next()) {                    //结果不为空时,利用查询结果构造 User 对象
                user=new User();
                user.setUserId(rs.getInt("userId"));
                user.setUserName(rs.getString("userName"));
                user.setPassword(rs.getString("password"));
                user.setGender(rs.getInt("gender"));
                user.setHead(rs.getString("head"));
                user.setRegTime(rs.getDate("regTime"));

            }
        } catch (Exception e) {e.printStackTrace();}
        finally {this.closeAll(conn, pstmt, rs); }
        return user;
    }
}
```

3.6.6 页面设计

1. 页面布局

留言板程序主要包括四个页面：登录页面 login.jsp、留言标题列表页面 index.jsp、查看留言详细内容页面 detail.jsp 和留言页面 post.jsp。另外，还包括三个辅助 JSP 文件 doLogin.jsp、doPost.jsp 和 logout.jsp。

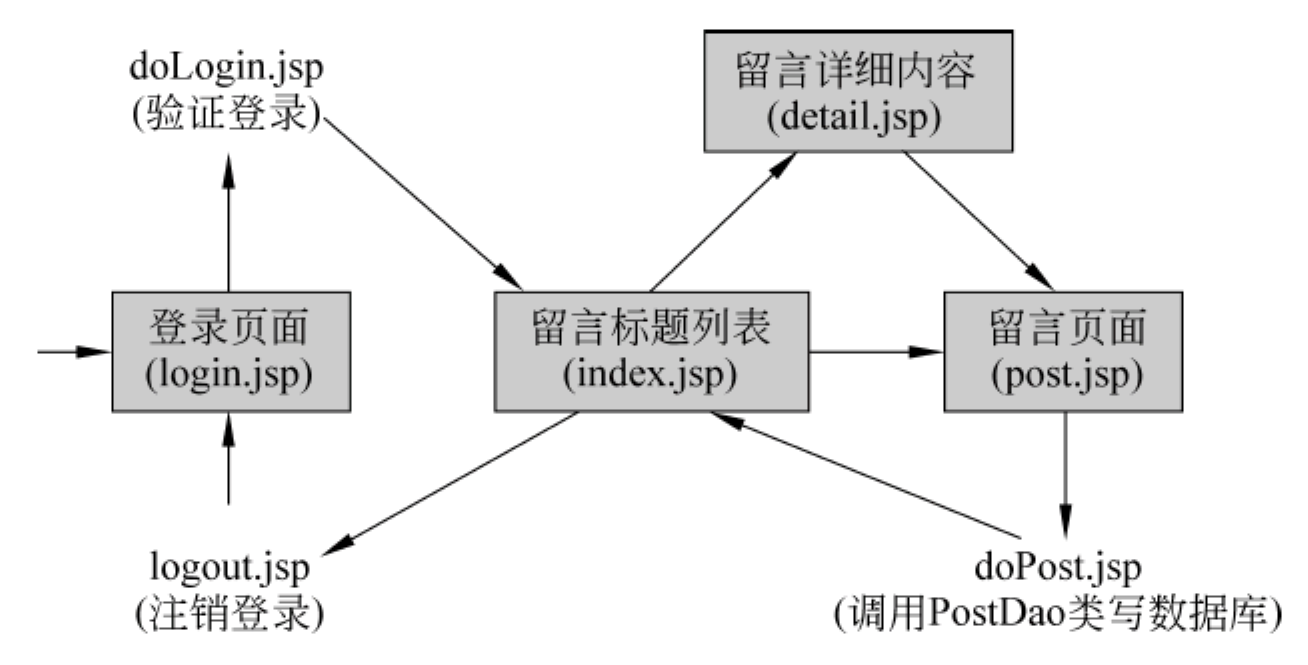


图 3-8　留言板程序页面关系

一个好的应用程序，页面的布局是非常重要的。基于表格的布局不再符合现代软件开发的潮流。为此，留言板程序采用 CSS+DIV 的方式进行布局。代码 3-23 给出了程序中用到的 CSS 文件。

代码 3-23　CSS 文件(main.css)

```
@CHARSET "UTF-8";
.page{margin: 20px,10px;width:960px;margin-left:-480px;left:50%;
    position:relative;}
.header{height:100px;margin-bottom:20px;background-image:
    url(../images/logo.png);background-repeat:no-repeat; }
.welcomePanel{right:10px;postion:relative;float:right;}
.mainDiv{margin-left:130px;margin-right:130px;width:800px;}
.col1{margin-top:10px;width:200px;text-align:right;float:left;}
.col2{margin-top:10px;width:500px;float:right;}
.leftalign{text-align:left}
.clear{clear:both;}
.userName{margin-left:0.5em;margin-right:0.5em;color:red;}
.ui-widget-header{background:#e78f08;color:#ffffff;font-weight:bold;
    height:30px;vertical-align:middle;}
.ui-widget-content{background:#eeeeee;color:#333333;}
```

由于每个页面的顶部都要显示 LOGO，都要对用户是否登录进行验证。从模块化设计的角度出发，把这部分内容放到一个 header.jsp 子页面中，然后通过 include 指令包含到其他页面中。header.jsp 的详细内容如代码 3-24 所示。

代码 3-24　header.jsp 子页面

```
<%@page language="java" import="jNotes.bean.*" pageEncoding="UTF-8"%>
<%
User user=null;
if(null!=session.getAttribute("user"))      /*检查用户是否登录*/
    user=(User)session.getAttribute("user");
else                                        /*用户未登录时,重定向到登录页面*/
    response.sendRedirect("login.jsp");
```

```
%>
<div class="header">
    <div class="welcomePanel">
        <%if(null!=user){ %>
            欢迎<span class="userName"><%=user.getUserName() %></span>回来
            <img alt="" src="images/split.jpg">
            <a href="logout.jsp">登出</a>
        <%} %>
    </div>
</div>
```

2 用户登录

用户登录部分由3个文件组成：login.jsp、doLogin.jsp和logout.jsp。

(1) login.jsp页面提供了用户登录界面，如代码3-25所示。

代码3-25 login.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%/*如果用户已经登录,直接转到index.jsp页面*/
if(null!=session.getAttribute("user"))
    response.sendRedirect("index.jsp");
%>
<html>
    <head>
        <title>欢迎登录JSP留言板</title>
        <link rel="stylesheet" type="text/css" href="style/main.css">
        <style><!--登录页面中用到的样式 -->
        .item{padding-left:380px;position:relative;padding-bottom:10px;}
        .label{width:360px;position:absolute;left:0;text-align:right;}
        </style>
    </head>
    <body>
        <div class="header"></div>
        <div class="ui-widget-header" style="text-align:center">登录</div>
        <div class="ui-widget-content">
        <form id="form1" action="doLogin.jsp" method="post">
        <div >
            <span class="label">用户名</span>
            <div class="item"><input name="userName" id="username"/></div>
        </div>
        <div>
            <span class="label">密码</span>
            <div class="item"><input name="password" type="password"/>
            </div>
        </div>
            <div class="item">
                <input type="submit" value="确认"/>
```

```
                <input type="reset" value="重写"/>
            </div>
        </form>
    </div>
  </body>
</html>
```

(2) doLogin.jsp 用于获取 Login.jsp 中传递过来的请求参数,并调用 UserDao 接口的 findUser()方法验证用户身份,如代码 3-26 所示。

代码 3-26 doLogin.jsp

```
<%@ page language="java" pageEncoding="UTF-8" import="jNotes.bean.*,jNotes.dao.*,jNotes.dao.
        impl.*" %>
<%
    /* 将获取的请求参数转码,以解决中文乱码问题 */
    String userName=new String(
          request.getParameter("userName").getBytes("ISO-8859-1"),
          "UTF-8");
    String password=new String(
          request.getParameter("password").getBytes("ISO-8859-1"),
          "UTF-8");
    /* 调用 UserDao 接口的 findUser()方法,验证用户身份 */
    UserDao userDao=new UserDaoImpl();
    User user=userDao.findUser(userName, password);
    if (null !=user) {  /* 通过身份验证,保存到 session,并重定向到 index.jsp*/
        session.setAttribute("user", user);
        response.sendRedirect("index.jsp");
    } else {  /* 未通过身份验证,重定向到 login.jsp*/
        response.sendRedirect("login.jsp");
    }
%>
```

(3) logout.jsp 注销登录

注销登录如代码 3-27 所示。

代码 3-27 logout.jsp

```
<%
    session.removeAttribute("user");          //从 session 中删除 user
    response.sendRedirect("login.jsp");       //重定向到登录页面
%>
```

图 3-9 给出了登录页面的界面。

3. 查看留言列表 index.jsp

index.jsp 使用 DAO 接口,获取所有的留言信息,并显示在页面中。代码 3-28 给出了 index.jsp 文件的内容,并附有详细的注释,在此不再赘述。



图 3-9 登录页面

代码 3-28 index.jsp

```
<%@ page language="java" pageEncoding="UTF-8"
  import="java.util.*,jNotes.dao.*,jNotes.dao.impl.*" %>
<%  /*获取所有留言列表*/
    NotesDao notesDao=new NotesDaoImpl();
    List<Notes> list=notesDao.getAllNotes();
%>
<html>
    <head>
        <title>留言列表</title>
            <link rel="stylesheet" type="text/css" href="style/main.css">
    </head>
    <body>
        <%@ include file="header.jsp" %>    <!--包含 header.jsp 子页面 -->
        <a href="post.jsp"><img src="images/book_write.png"/></a>
        <div class=" ui-widget-header" style="text-align:center;">留言列表
        </div>
        <div class="col1">留言人</div>
        <div class="col2">主题 </div>
        <%  /*迭代留言列表,利用动态生成的 div 显示留言人和留言主题*/
            Iterator<Notes> it=list.iterator();
            while(it.hasNext()){
                Notes note=it.next();
        %>
            <div class="clear"></div>
        <div class="col1">
            <%=note.getUser().getUserName() %><!--显示留言人 -->
        </div>
        <div class="col2">
        <a href="detail.jsp?noteId=<%=note.getNoteId() %>">
            <%=note.getTitle() %><!--动态构造每条留言的 URL -->
        </div>
```

```
        <%} %>
    </body>
</html>
```

4. 查看留言内容

detail.jsp 根据 index.jsp 页面传过来的留言 ID，调用 DAO 接口获取留言的详细信息，并显示在页面中。代码 3-29 给出了 detail.jsp 文件的内容，并附有详细的注释，在此不再赘述。

代码 3-29　detail.jsp

```
<%@ page language="java" import="jNotes.dao.*,jNotes.dao.impl.*"
    pageEncoding="UTF-8"%>
<%/* 从请求参数中获取留言的 ID -->
int noteId=Integer.parseInt(
    request.getParameter("noteId").toString());
/* 根据留言 ID,获取留言的详细信息 */
Notes note=DaoFactory.getNotesDao().getNoteById(noteId);
%>
<html>
    <head>
        <title>留言内容</title>
    <link rel="stylesheet" type="text/css" href="style/main.css">
    </head>
    <body>
        <%@ include file="header.jsp" %><!--包含 header.jsp 子页面 -->
        <a href="post.jsp"><img src="images/book_write.png"/></a>
        <div class="ui-widget-header" style="text-align:center;">留言内容
        </div>
        <div class="col1">留言人</div>
            <div class="col2">内容 </div>
            <div class="clear"></div><!--清除 div 浮动效果-->
            <div class="col1 leftalign"><!--显示留言人信息和留言内容 -->
            <img src="images/head/<%=note.getUser().getHead() %>"/><br/>
                用户名:<%=note.getUser().getUserName() %><br/>
                注册时间:<%=note.getUser().getRegTime() %>
            </div>
        <div class="col2"><%=note.getContent() %></div>
    </body>
</html>
```

图 3-10 给出了查看留言内容页面的运行效果。

5. 留言

post.jsp 为用户创建新的留言提供了接口。代码 3-30 给出了 post.jsp 文件的内容，并附有详细的注释，在此不再赘述。



图 3-10　留言内容

代码 3-30　post.jsp

```
<%@ page language="java" import="jNotes.dao.*,jNotes.dao.impl.*"
    pageEncoding="UTF-8"%>
<html>
    <head>
        <title>新留言</title>
        <link rel="stylesheet" type="text/css" href="style/main.css">
    </head>
    <body>
        <%@ include file="header.jsp" %><!-- 包含 header.jsp 子页面 -->
        <div class="mainDiv">
        <div class="ui-widget-header" style="text-align:center;">新留言
        </div>
            <form name="form1" action="doPost.jsp" type="post">
                <div class="col1">标题</div>
                <div class="col2"><input name="title" size="50" /></div>
                <div class="col1">内容</div>
                <div class="col2">
                <textarea rows="20" cols="60" name="content"></textarea>
                            <br/>(少于 200 字)
                </div>
                <div class="col2">
                    <input type="submit" value="确定" />
                    <input type="reset" />
                </div>
            </form>
        </div>
    </body>
</html>
```

图 3-11 给出了创建新留言页面的运行效果。



图 3-11　留言页面

doPost.jsp 负责接收 post.jsp 传递过来的参数，调用 DAO 接口中的相应方法将留言的内容保存到数据表中，最后重定向到留言列表页面 index.jsp，如代码 3-31 所示。

代码 3-31　doPost.jsp

```
<%@ page language="java" pageEncoding="UTF-8"
    import="jNotes.bean.*,jNotes.dao.*,jNotes.dao.impl.*"%>
<%
    /* 接收请求参数,并解决中文乱码 */
    String title=new String(request.getParameter("title")
                .getBytes("ISO-8859-1"),"UTF-8");
    String content=new String(request.getParameter("content")
                .getBytes("ISO-8859-1"),"UTF-8");
    Notes note=new Notes();
    note.setContent(content);
    note.setTitle(title);
    note.setUser((User)session.getAttribute("user"));
    NotesDao noteDao=new NotesDaoImpl();
    noteDao.addNote(note);          /* 调用 NotesDao 接口 */
    response.sendRedirect("index.jsp");
%>
```

同 步 训 练

1. 利用 3.6 节给出的 DAO 接口，为留言板程序增加用户注册功能。
2. 仿照 3.6 节给出的留言板程序样例，设计一个站内短信系统。要求如下：
(1) 设计合理的数据库结构；
(2) 编写 PO 类和 DAO 类；
(3) 实现用户注册、登录、站内短信查看、回复、写信以及注册用户查询等功能。



Struts 2 基础

4.1 认识 Struts 2

Struts 2 是一个开源的 Web 开发框架，它由两部分组成：XWork 2 和 Struts 2。XWork 是一个命令模式框架，它提供了 Struts 2 框架的核心功能：IoC（控制反转）容器、强大的表达式语言（OGNL）、数据类型转换、输入验证和可插入的配置。XWork 框架的核心概念包括 action、拦截器（interceptor）和 result。Struts 2 扩展了这些概念的基础实现，用于支持 Web 应用程序的开发。

Struts 2 框架具有如下特点。

（1）基于 Action 的框架。Struts 2 将处理用户请求的逻辑封装到了 Action 类中，并通过使用 Struts. xml 完成客户请求和 Action 类的映射关系，避免了纯 JSP 编程中由于 HTML 标签和 JSP 代码混合所带来的难以维护的问题。

（2）Struts 2 采用 XML 文件作为映射、校验、类型转换和国际化处理时的配置，提高了应用程序的可维护性。

（3）Struts 2 整合了 OGNL（Object-Graph Navigation Language）。作为一种功能强大的表达式语言，OGNL 提供了一种简单一致的表达式语法，用于存取对象的每个属性，调用对象的方法，实现字段类型转化等功能。

（4）Struts 2 提供了自己的标签库，通过与 OGNL 联合使用，使得用户能够更容易地编写表示层。

（5）Struts 2 能够很方便地与其他流行框架结合使用。例如，与 Spring 结合使用，可以充分利用 Spring 的 IoC 技术，促进系统的松耦合；与 SiteMesh 结合使用，能够帮助网站开发人员较容易地实现页面中动态内容和静态装饰外观的分离；与 Hibernate 或 iBatis 结合使用，可以减少程序员进行持久化处理的工作量。

（6）Struts 2 拥有由积极活跃的开发人员与用户组成的成熟社区，是当前最为流行的 Web 开发框架技术之一。

4.2 创建 Struts 2 应用程序

4.2.1 Struts 2 开发步骤

要使用 Struts 2 进行 Web 开发，首先要从 http://struts. apache. org/download. cgi

下载 Struts 2 开发包。作者在编写本书时，Struts 2 的最新版本为 2.3.4。Apache 为开发者提供了很多压缩包，建议初学者下载完整版（Full Distribution），即 struts-2.3.4-all.zip。该压缩包中包含 Struts 2 的文档、示例程序、源文件和核心类库以及 Struts 2 所依赖的类库等。

解压 struts 2.3.4-all.zip 后，可以得到如下目录结构。

（1）apps：该文件夹包含基于 Struts 2 的示例程序。

（2）docs：该文件夹包含 Struts 2 的相关文档，包括 Struts 2 快速入门、Struts 2 的文档以及 API 文档等内容。

（3）lib：该文件夹包含 Struts 2 框架的核心类库，以及与 Struts 2 相关的所有第三方插件类库。

（4）src：该文件夹包含 Struts 2 框架的全部源代码。

准备好 Struts 2 开发包之后，就可以进行 Struts 2 的开发工作。下面以留言板程序的登录部分为例，讲解利用 Struts 2 进行 Web 应用程序开发的流程。

首先，打开 myEclipse，创建一个 Web Project。然后，按照下述步骤进行配置和开发。

1. 准备类库

不同的开发需求使用到的类库是不一样的。以 Struts 2-2.3.4 为例，至少应该包含以下 7 个 JAR 包。

（1）xwork-core-2.2.1.jar：XWork 类库，Struts 2 框架的基础。

（2）Struts 2-core-2.2.1.jar：Struts 2 框架的核心库。

（3）commons-fileupload-1.2.1.jar：Struts 2 用于上传文件的类库，无论应用程序中是否用到了文件上传功能，必须包含本类库。

（4）commons-io-1.3.2.jar：通用输入输出类库。

（5）ognl-3.0.jar：对象图导航语言，Struts 2 使用的一种表达式语言。

（6）freemarker-2.3.16.jar：Struts 2 的 UI 标签的模板是使用 FreeMarker 编写的，无论在应用程序中是否用到了 Struts 2 标签，必须包含本类库。

（7）javassist-3.7.ga.jar：用于动态编辑、生成 Java 字节码的类库。

将上述类库复制到 Web 应用的 WEB-INF/lib 文件夹中。

2. 配置 FilterDispatcher

首先，在 web.xml 文件中配置 Struts 2 的核心控制器 StrutsPrepareAndExecuteFilter。该控制器负责加载 Struts 2 的相关配置文件，用来拦截客户端请求，并把请求转发到相应的 Action 类来处理，如代码 4-1 所示。

代码 4-1　web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
    <display-name>Struts Blank</display-name>
    <filter>
        <filter-name>struts2</filter-name>
```

```
            <filter-class>
                org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
            </filter-class>
        </filter>
        <filter-mapping>
            <filter-name>struts2</filter-name>
            <url-pattern>/*</url-pattern>
        </filter-mapping>
        <welcome-file-list>
            <welcome-file>index.html</welcome-file>
        </welcome-file-list>
    </web-app>
```

web.xml 中关于配置这一步是固定的，大家可以在做好一个 web.xml 文件后保存起来，以后再编写 Struts 2 应用时直接将其复制过去即可。

注意：核心控制器 StrutsPrepareAndExecuteFilter 的本质就是一个 Servlet 过滤器，这里把 url-pattern 配置成/*，意思是让控制器拦截所有的用户请求。如果在 Web 应用程序中还需要编写部分 Servlet 类，应该将 url-pattern 配置成.action。此时，控制器将只拦截 url 扩展名为.action 的用户请求，而不会处理 Servlet 请求。

3. 创建业务控制器 LoginAction

Struts 2 的业务控制器就是一个 POJO(Plain Old Java Objects，简单的 Java 对象)。该 POJO 类包含多个属性，用于封装用户的请求参数和需要呈现给客户端的数据，并提供了一个返回类型为 String 的公共方法用于处理用户的请求。默认时，该方法为 execute()，如代码 4-2 所示。

代码 4-2　LoginAction.java

```
package example.action;
public class LoginAction {
    private String userName;            //定义登录名称属性
    private String password;            //定义登录密码属性
        /* 省略了 get 和 set 方法的代码，请读者自行添加 */

    public String execute() {           //登录验证
        if ("zhangsan".equals(username)&&"123".equals(password))
            return "success";           //登录成功返回 success
        else
            return "login";             //登录失败返回 login
    }
}
```

4. 创建视图页面

编写留言板程序的登录页面 login.jsp，如代码 4-3 所示。注意，login.jsp 中 form 的 action 属性的取值为 login.action。如果在 web.xml 中，核心控制器拦截的 URL 后缀为/*，可以省略这里的扩展名.action。

代码 4-3　login.jsp

```
<!-- login.jsp 文件源码 -->
<%@ page contentType="text/html; charset=utf-8" language="java"%>
<html>
    <head>
        <title>登录</title>
    </head>
    <body>
        <form action="login.action" method="post">
            用户名：<input type="text" name="userName" /><br />
            密   码：<input type="password" name="password" /><br />
            <input type="submit" value="登录" />
        </form>
    </body>
</html>
```

5. 创建欢迎页面 welcome.jsp

编写登录成功后的欢迎页面 welcome.jsp，如代码 4-4 所示。

代码 4-4　欢迎页面 welcome.jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<html>
    <head>
        <title>登录成功</title>
    </head>
    <body>
        <h2>
            <s:property value="userName"/>您好，欢迎登录系统！
        </h2>
    </body>
</html>
```

6. 配置 LoginAction

为了让 Action 能处理用户的请求，还需要在 struts.xml 文件中配置 Action。struts.xml 文件存放在 classes 路径下，在利用 myEclipse 开发时可以直接放在 src 目录下。在 struts.xml 文件中配置 Action 时，用 name 属性定义该 Action 的名称，用 class 定义这个 Action 的实际实现类。由于当 Action 处理完客户端请求后返回一个字符串，每个字符串都将对应一个视图。因此，配置 Action 时，需要为每个 Action 指定 result 元素，定义 Action 返回字符串对应的视图。

在代码 4-5 中定义了一个名字为 login 的 Action，其实现类为 example.action.LoginAction。该 Action 将负责处理 URL 为 login.action 的用户请求。处理时，Action 将调用它的 execute()方法处理用户请求。如果 execute()方法返回结果字符串 success，则请求将被转发到/success.jsp 页面；如果 execute()方法返回 login，则请求被转发到 login.jsp 页面。

代码 4-5 struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>
    <package name="default" namespace="/" extends="struts-default">
        <action name="login" class="example.action.LoginAction">
            <result name="success">/welcome.jsp</result>
            <result name="login">/login.jsp</result>
        </action>
    </package>
</struts>
```

现在，在 myEclipse 中将 notes 发布到 tomcat 下，并启动 tomcat 服务。打开浏览器访问 http://localhost:8080/notes/，在输入完用户名和密码后，可以看到熟悉的留言板主页面，如图 4-1 所示。



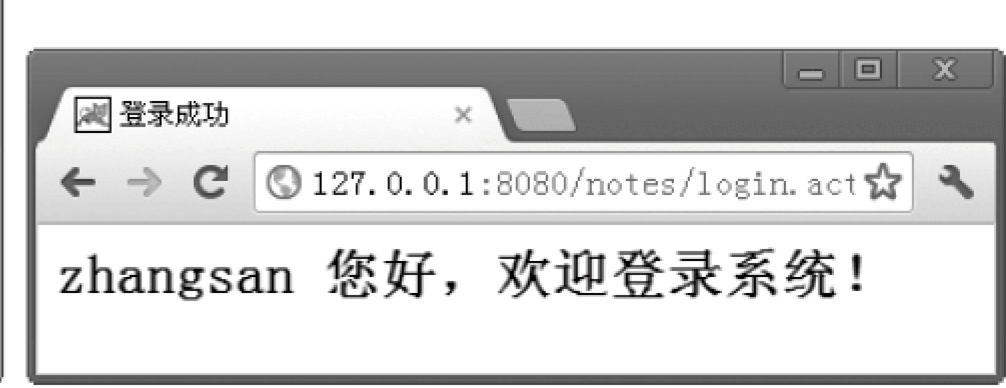


图 4-1 留言板登录页面和登录成功页面

下面分析该程序的工作流程。

(1) Struts 2 容器收到 login.action 请求，从 web.xml 获取核心过滤器。org.apache.Struts 2.dispatcher.FilterDispatcher 是所有应用(包括 *.action)的入口点。

(2) Struts 2 在 struts.xml 中找到 loginAction 类(Action)，然后调用 setUserName()和 setPassword()方法分别为属性 userName 和 password 赋值，并调用 loginAction 类的 execute()方法。

(3) execute()方法对请求进行处理，并根据处理返回不同的结果代码，Struts 2 收到结果代码后，按照映射关系，把相应的 Web 页面返回给客户端。

(4) 如果要返回给客户端的是页面 welcome.jsp，则在返回前，<s:property value="userName" />会调用 LoginAction 类 getUserName()方法，获取属性 userName 的值，并显示在页面上。

提示：初学者在编写和执行第一个 Struts 2 应用程序时，会遇到各种各样的错误。为保证程序顺利执行，在利用 myEclipse 开发 Struts 2 应用时，必须注意以下几点。

(1) 确保必需的 Struts 2 的类库复制到 WebRoot\WEB-INF\lib 目录中。

(2) Web.xml 应该放在 WebRoot\WEB-INF 目录下，并且正确配置核心控制器。

(3) 表单中各个 input 标签的 name 取值必须和 Action 类中属性的名字完全一致，并且为 Action 类中的每个属性都编写了 Get 和 Set 方法。

(4) 如果结果视图(.jsp 文件)中需要显示 Action 中的属性值，则必须保证结果视图中所用属性名和 Action 类中一致。

(5) struts.xml 应该放在 src 目录下，正确配置 Action 和 Action 对应的类、结果类型和结果映射视图。

(6) struts 2-2.3.4-all.zip 的示例文档中有一个 struts 2-blank-2.3.4.war，将其解压之后得到空的 Struts 2 工程。初学者可以从该工程中将所需的 jar 类库、web.xml 和 struts.xml 复制到自己的工程中。

4.2.2　扩展 ActionSupport 类

Action 类是 Struts 2 的核心内容。应用程序能够完成的每一个功能都对应 Action 或者 Action 中的一个方法。客户端通过表单或者 URL 参数提交的数据被 Struts 2 传递给了 Action 对象，同时在 Struts 2 处理 Action 结果映射时，将 Action 对象中的属性值响应给用户。

创建 Action 类是 Struts 2 中最重要的任务。Action 类就是一个普通的 POJO 类，每个属性都需要编写 get 方法(向 Action 对象的属性传值时调用)和 set 方法(Action 对象向 JSP 页面或其他 Action 传值时调用)。

每个 Action 类必须包含一个无参的构造函数。正常情况下，一个 Java 类如果没有编写任何构造函数，编译程序会自动生成一个无参的构造函数。如果编写了带有参数的构造函数，就必须编写一个无参的构造函数(即使这个函数什么也不做)。

每个 Action 类至少提供一个 public 类型的方法供 Struts 2 在执行 Action 时调用，该方法的返回类型必须为 String 类型。默认时，Struts 2 会自动调用 execute()方法。

在代码 4-2 中，利用一个普通的 POJO 类充当了留言板登录程序的 Action 处理类。在 Struts 2 的开发包中提供了一个 com.opensymphony.xwork2.Action 接口。代码 4-6 给出了 Action 接口的源代码。接口中定义了一个默认的业务逻辑方法 execute()和五个常量，这五个常量就是在日常开发中业务逻辑方法返回的字符串。程序员在编写 Action 类时可以直接实现 Action 接口，但更多的是扩展 Action 接口的实现类 com.opensymphony.xwork2.ActionSupport。

代码 4-6　Action 接口

```
package com.opensymphony.xwork2;
public interface Action {
    public static final String SUCCESS= "success";
    public static final String NONE= "none";
    public static final String ERROR= "error";
    public static final String INPUT= "input";
    public static final String LOGIN= "login";
    public String execute() throws Exception;
}
```

ActionSupport 类在实现 Action 接口的同时，实现了 ValidationAware 和 LocaleProvider 等。扩展 ActionSupport 类，Action 类能够很方便地完成数据验证、国际化等工作。代码 4-7 通过扩展 ActionSupport 类给出了代码 4-2 的另一个版本 LoginAction2.java。

代码 4-7　LoginAction2.java

```
package example.action;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction2 extends ActionSupport {
    private String userName;                    //定义登录名称属性
    private String password;                    //定义登录密码属性
        /* 省略了 get 和 set 方法的代码,请读者自行添加 */

    public String execute() {                   //登录验证
        if("zhangsan".equals(userName)&&"123".equals(password))
            return SUCESS;                      //登录成功返回 success
        else
            return LOGIN;                       //登录失败返回 login
    }
}
```

注意：由于 ActionSupport 是 Action 接口的实现类，因此代码 4-7 中可以直接使用 SUCCESS 和 LOGIN 等结果代码。

4.3 接收用户输入

在开发 Web 应用程序的过程中，不可避免地会遇到处理用户输入数据的问题。Struts 2 提供了几种方法，使用户能够很方便地将请求参数绑定到 Action 属性中。

4.3.1 属性驱动

在 Struts 2 中，可以直接使用 Action 的属性来接收用户的输入。代码 4-8 给出了一个用于信息维护的页面 userInfor.jsp。表单中包含一个文本框 userName，两个单选按钮 gender 和三个复选框 favorite。注意，处理表单的 Action 属性的值为 userAction.action?id=1，包含了一个请求参数 url。

代码 4-8　userInfor.jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
<head><title>个人信息维护</title></head>
<body>
    <form name="regForm" action="userAction.action?id=1" method="post">
    用户名：
        <input type="text" maxlength="20" size="40" name="userName" /><br/>
    性别：
```

```
        女<input type="radio" name="gender" value="1" />
        男<input type="radio" name="gender" value="2" checked /><br/>
    喜好的运动:
        <input type="checkbox" name="favorite" value="篮球" checked />篮球
        <input type="checkbox" name="favorite" value="足球" />足球
        <input type="checkbox" name="favorite" value="爬山" />爬山<br/>
        <input type="submit" value="确认" />
        <input type="reset" value="重置" />
    </form>
</body>
</html>
```

下面编写一个 Action 类 UserAction.java，用于接收代码 4-8 给出的页面的提交数据。

注意： 当采用 Action 属性接收输入时，Action 类中必须定义与 url 参数名以及 Input 标签的 name 同名的属性，并为每个属性定义 set 方法。对于表单中的 favorite，由于对应一组输入值，在 Action 类中需要使用 List 或数组来接收。出于演示的目的，UserAction.java 在接收到用户输入后，直接将数据打印到控制台上。

UserAction.java 的内容如代码 4-9 所示。这里采用 List 接收一组 checkbox 的输入。

代码 4-9　UserAction.java

```
package example.action;
import java.util.Iterator;
import java.util.List;
import com.opensymphony.xwork2.ActionSupport;
public class UserAction extends ActionSupport {
    private String userName;
    private int gender;
    private List<String> favorite;    //用于接收 checkbox 的值
    private int id;                   //用于接收 url 请求参数 id 的值
    //省略了 userName,gender,favorite 和 id 的 get 和 set 方法
    //...
    public String execute() throws Exception {
        System.out.println("userName="+userName);
        System.out.println("gender="+gender);
        System.out.println("id="+id);
        System.out.print("favorite=");
        Iterator<String> it=favorite.iterator();
        while(it.hasNext()){
            System.out.print(it.next()+" ");
        }
        return SUCCESS;
    }
}
```

注意：UserAction.java 中的属性 gender 和 id 的类型为 int。在基于 HTTP 的协议中，客户端和服务器之间传输的数据都是字符串数据。Struts 2 提供了自动类型转换功能，能够将客户端传送来的数据自动转化为正确的类型。这一部分内容将在后面的章节中讲述。

在 Struts 2 接收到表单提交请求后，会自动调用 UseAction 对象的各个 set 方法，将输入数据绑定到同名的属性上。

4.3.2 模型驱动

尽管利用 Action 属性能够完成对用户输入数据的接收，但是在实际的开发中，往往会把用户输入的信息封装在一个 Model 中，此时需要使用领域模型来接收用户输入。

代码 4-10 为经过改写的用户登录页面的代码，请注意 input 标签的取值。

代码 4-10 登录表单

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
    <head><title>登录</title></head>
    <body>
        <h2>请您登录</h2>
        <form name="form1" action="login3.action" method="post">
            用户名：<input type="text" name="user.userName"><br>
            口令：<input type="password" name="user.password"><br>
            <input type="submit" value="提交">
            <input type="reset" value="重置">
        </form>
    </body>
</html>
```

代码 4-11 中定义了一个 JavaBean 类 User.java，然后在 LoginAction3.java 中利用一个 User 对象 user 处理用户输入。

代码 4-11 利用领域对象接收用户输入数据

```
/*** User.java ***/
package example.Model;
public class User {
    private String username;            //用户名
    private String password;            //密码
    //省略了 username、password 属性的 get 和 set 方法
}

/*** LoginAction3.java ***/
package example.action;
import example.Model.User;
import com.opensymphony.xwork2.Action;
public class LoginAction3 implements Action {
    private User user;                  //定义领域对象 user
```

```
        public User getUser() {
            return user;
        }
        public void setUser(User user) {
            this.user=user;
        }
        public String execute() throws Exception {
            if ("zhangsan".equals(user.getUserName())
               &&"123".equals(user.getPassword()))
                return SUCCESS;              //登录成功返回 SUCCESS
            else
                return LOGIN;                //登录失败返回 LOGIN
        }
    }
```

在这个例子中，表单中包括了两个 name 属性的取值分别为 user. userName 和 user. password 的 input 标签。这里的 user 与 LoginAction3. java 中的领域属性同名，username 和 password 是领域属性 user 的属性名。根据 Struts 2 的数据绑定机制，在向 Action 传递数据时，user. userName 等同于调用 getUser(). setUserName()。

请注意，LoginAction3. java 中的 user 在定义后并没有实例化，但在执行语句

```
"zhangsan".equals(user.getUserName())
```

时，程序并没有抛出 NullPointerException 异常，这是因为 Struts 2 提供了自动实例化对象特性。当 Struts 2 发现 User 对象为空时，会自动调用 User 类的无参构造函数构造一个 User 实例。

4.3.3　实现 ModelDriven

在 4. 3. 2 小节给出的利用领域对象接收用户输入数据的方法中，要求 JSP 页面中 input 标签的 name 属性必须使用"领域对象名. 属性名"的格式命名。Struts 2 提供了另外一种模型驱动的方法，即让 Action 类实现 com. opensymphony. xwork2. ModelDriven 接口，该接口中只定义了一个方法。

```
public T getModel()
```

使用 getModel()方法来通知 Struts 2 要绑定的属性类型。与 4. 3. 2 小节给出的模型驱动方法不同的是，领域对象在声明时一定要实例化，但是不需要提供 get 方法和 set 方法。代码 4-12 为实现了 ModelDriven 的 Action 类。

代码 4-12　实现了 ModelDriven 的 Action 类

```
package example.action;
import example.Model.User;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;
public class LoginAction4 extends ActionSupport
```

```
        implements ModelDriven<User> {
    private User user=new User();              //user一定要实例化
    public String execute() throws Exception {
        if ("zhangsan".equals(user.getUserName())
                && "123".equals(user.getPassword())){
            return SUCCESS;                    //登录成功返回 SUCCESS
        }
        else
            return LOGIN;                      //登录失败返回 LOGIN
    }
    public User getModel() {
        return user;
    }
}
```

在实现了 ModelDriven 接口之后，输入表单中 input 标签的 name 命名时就不必加上 user 前缀了，只需要与 User 类的 userName 和 password 属性同名即可，具体代码可以参见代码 4-3。

Struts 2 之所以能够完成输入请求和 Action 类之间的数据绑定，完全是依靠 params、prepare 和 ModelDriven 拦截器发挥的作用。关于拦截器，将在后面的章节中讲述。

4.4 跟踪用户状态

在纯 JSP 编程时，利用 request、application 和 session 等内置对象可以获得用户的请求信息，实现用户响应和完成会话跟踪等工作。在 Servlet 编程时，也可以通过 HttpServletRequest、HttpSession 和 ServletContext 等完成同样的工作。尽管利用 Struts 2 框架提供的数据绑定等功能能够方便地完成上述用户请求和响应处理，但在有些情况下仍然需要在 Action 中访问上述几种 Servlet 对象。例如，当一个用户登录成功后，需要将用户信息添加到 session 中。

4.4.1 利用非 IoC 方式跟踪用户状态

IoC(Inversion of Control，控制反转)就是将应用系统中原来由程序控制“对象之间的关系”转交给由外部容器来实现控制。利用非 IoC 方式跟踪用户状态，就是采用直接编写程序代码的方式控制生成与状态跟踪有关的对象实例。

Struts 2 提供了两种方法实现利用非 IoC 方式跟踪用户状态：一种方法是利用 ActionContext；另一种方法是利用 ServletActionContext。

1. ActionContext

ActionContext 是一个容器，主要存储 request、session、application 和 parameters 等与 Action 执行有关的上下文信息。在每次执行 Action 之前都会创建新的 ActionContext。ActionContext 是线程安全的，也就是说，在同一个线程里，ActionContext 里的属性是唯一

的。这样,用户的 Action 可以在多线程中使用。

ActionContext 类定义了一系列方法,用于访问封装了 HttpServletRequest、HttpServletResponse、HttpSession 和 ServletContext 等信息的 Map 对象。

(1) public Map<String, Object>getSession():返回一个 Map 对象。该对象模拟了 HttpSession 实例。

(2) public void setSession(Map<String, Object>session):设置 session。

(3) public Map<String, Object>getApplication():返回一个 Map 对象。该对象模拟了 ServletContext 实例。

(4) public void setApplication(Map<String, Object>application):设置 Action 的 application 上下文。

(5) public Map<String, Object>getParameters():返回一个 Map 对象。该对象封装了所有 HttpServletRequest 请求参数。

(6) public Object get(String key):从 ActionContext 中取出一个值,类似于调用 HttpServletRequest 的 getParameter()方法。当 key 的值取 request 时,相当于取出一个 HttpServletRequest 实例。

从 ActionContext 提供的方法可以看出,Struts 2 把 ServletContext 等信息封装成了一个个 Map 对象,key 是标识 request、session 等的字符串,值是其对应的对象。

ActionContext 是一个线程的本地变量,这意味着,不同的 Action 之间不会共享 ActionContext,所以也不用考虑线程安全问题。

接下来,改写代码 4-2 中给出的用户登录 Action,利用 ActionContext 增加对用户状态的跟踪,如代码 4-13 所示。

代码 4-13　LoginAction.jsp(利用 ActionContext 访问 request 等)

```
package example.action;
import java.util.Map;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class LoginAction extends ActionSupport {
    private String userName;
    private String password;
    /* 省略了 getter 和 setter 方法的代码,请读者自行添加 */

    public String execute() {
            ActionContext context=ActionContext.getContext();
            Map session=context.getSession();
            session.put("user",userName);

            Map request=(Map) context.get("request");
            request.put("welcome", "您好,欢迎登录系统!");

            Map application=context.getApplication();
            Integer cnt=(Integer)application.get("count");
            if(null==cnt)
```

```
                    cnt=1;
                else
                    cnt++;
                application.put("count", cnt);
                return SUCCESS;
        }
    }
```

LoginAction 类的成功页面将映射到 welcome.jsp，将利用 Struts 2 提供的 property 标签将保存在 session、request 和 application 中的数据取出来。有关 Struts 2 标签的知识，将在后续章节中讲述。welcome.jsp 的内容如代码 4-14 所示。

代码 4-14　welcome.jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<html>
    <head>
        <title>登录成功</title>
    </head>
    <body>
        <ul>
            <li>
                下面的 zhangsan 是利用 action 属性传递过来的
                <h2><s:property value="userName"/>
                    <s:property value="#request.welcome"/></h2>
            </li>
            <li>
                下面的 zhangsan 是利用 request 对象传递过来的
                <h2><s:property value="#request.userName"/>,您是第
                    <s:property value="#application.count"/>个访问者。</h2>
            </li>
            <li>
                下面的 zhangsan 是利用 session 对象传递过来的
                <h2><s:property value="#session.user"/>,欢迎您!</h2>
            </li>
        </ul>
    </body>
</html>
```

测试修改之后的留言板登录程序。图 4-2 给出了成功登录后的页面。welcome.jsp 显示了三次 zhangsan：第一个 zhangsan 是通过绑定 LoginAction 类的属性 userName 得到的；第二个 zhangsan 是利用 request 对象获取的，由于在 ActionContext 中封装了一个 HttpServletRequest 的 Map 对象，所有绑定到 Action 类的属性的数据同样会传递给该 Map 对象；第三个 zhangsan 显示的是保存在 session 中的数据。

需要注意的是，最好不要在 Action 类的构造函数中访问 ActionContext，因为此时部分数据尚未准备完成。



图 4-2　welcome.jsp 运行界面

2. ServletActionContext

类 org.apache.Struts 2.ServletActionContext 继承了 ActionContext 类，它提供了直接对 Servlet 相关对象访问的功能。利用 ServletActionContext 可以直接访问如下 Servlet 的对象。

(1) javax.servlet.http.HttpServletRequest：HttpServlet 请求对象。

(2) javax.servlet.http.HttpServletResponse：HttpServlet 相应对象。

(3) javax.servlet.ServletContext：Servlet 上下文环境。

(4) javax.servlet.ServletConfig：Servlet 配置对象。

(5) javax.servlet.jsp.PageContext：HTTP 页面上下文。

程序员可以利用 ServletActionContext 类提供的静态方法获得与用户跟踪相关的 Servlet 对象。

(1) 获得 ServletContext 对象：

```
HttpServletContext context=ServletActionContext.getServletContext();
```

JSP 中的 application 内置对象是 javax.servlet.ServletContext 接口的一个实例，因此这里的 context 就相当于前面学过的 application 对象，程序员可以直接利用 context 通过 getAttribute()方法和 setAttribute()方法访问需要在 Application 一级保存的数据。

(2) 获得 HttpServletRequest 对象：

```
HttpServletRequest request=ServletActionContext.getRequest();
```

(3) 获得 HttpSession 对象：

```
HttpSession session=ServletActionContext.getRequest().getSession();
```

类 ServletActionContext 没有提供直接获得 HttpSession 对象的方法，HttpSession 对象可以通过 HttpServletRequest 对象来获得。

请读者自行利用 ServletActionContext 类改写本节中的程序。

4.4.2 利用 IoC 方式跟踪用户状态

所谓利用 IoC 方式跟踪用户状态，是指程序员不必在 Action 类中通过 ActionContext 类或 ServletActionContext 类手工获取与 request 或是 session 有关的 Map 对象或 Servlet 实例，而是让 Struts 2 框架在运行时向 Action 中注入 session、request 和 application。

与非 IoC 方式类似，Struts 2 也提供了两种 Servlet 对象注入方式：一种是注入包装成 Map 对的 Servlet；另一种方式是注入真实的 Servlet 对象。

Struts 2 提供了三个接口用于访问 session、request 和 application。

(1) org.apache.struts2.interceptor.SessionAware。Struts 2 利用该接口向 Action 实例注入 session 的 Map 对象。实现该接口时，必须实现如下方法：

```
public void setSession(Map<String,Object> session)
```

(2) org.apache.struts2.interceptor.RequestAware。Struts 2 利用该接口向 Action 实例注入 request 的 Map 对象。实现该接口时，必须实现如下方法：

```
public void setRequest(Map<String,Object> request)
```

(3) org.apache.struts2.interceptor.ApplicationAware。Struts 2 利用该接口，向 Action 实例注入 application 的 Map 对象。实现该接口时，必须实现如下方法：

```
public void setApplication(Map<String,Object> application)
```

在代码 4-15 中，LoginAction 类通过实现上述三个接口来跟踪用户状态。

代码 4-15 LoginAction.java(实现 SessionAware 等接口)

```
package example.action;

import java.util.Map;

import org.apache.struts2.interceptor.ApplicationAware;
import org.apache.struts2.interceptor.RequestAware;
import org.apache.struts2.interceptor.SessionAware;
import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport
        implements SessionAware,RequestAware,ApplicationAware {
    private String userName;
    private String password;
    private Map session;
    private Map request;
    private Map application;

    /* 省略了 getter 和 setter 方法的代码,请读者自行添加 */

    public void setSession(Map session) {
```

```
            this.session= session;
        }
        public void setRequest(Map request) {
            this.request= request;
        }
        public void setApplication(Map application) {
            this.application= application;
        }

        public String execute() {
                session.put("user",userName);
                request.put("welcome", "您好,欢迎登录系统!");

                Integer cnt= (Integer)application.get("count");
                if(null==cnt)
                    cnt=1;
                else
                    cnt++;
                application.put("count", cnt);

                return SUCCESS;

        }
    }
```

成功返回页面与 4.4 节中是一样的,这里不再赘述。

如果希望 Struts 2 框架在 Action 实例运行时注入真实的 Servlet 对象,需要让 Action 类实现如下接口。

(1) org.apache.struts2.interceptor.ServletRequestAware。Struts 2 框架利用该接口向 Action 类注入 HttpServletRequest 对象。实现该接口时,需要实现如下方法:

```
void setServletRequest(HttpServletRequest request)
```

(2) org.apache.struts2.interceptor.ServletContextAware。Struts 2 框架利用该接口向 Action 类注入 ServletContext 对象。实现该接口时,需要实现如下方法:

```
void setServletContext(ServletContext context)
```

请读者自行利用上述两个接口改写代码 4-14。

4.5　MyEclipse 提供的 Struts 2 添加向导

在开发基于 Struts 2 的 Web 应用程序时,需要为应用程序添加 Struts 2 的支持包,在 Struts 2 中配置核心过滤器,配置 Struts.xml 文件,这些工作除了手工完成外,可以使用 MyEclipse 提供的 Add Struts Capabilities 向导来进行。

依次执行 MyEclipse→Project Capabilities→Add Struts Capabilities 菜单命令，打开添加 Struts 支持向导 Add Struts Capabilities，如图 4-3 所示。

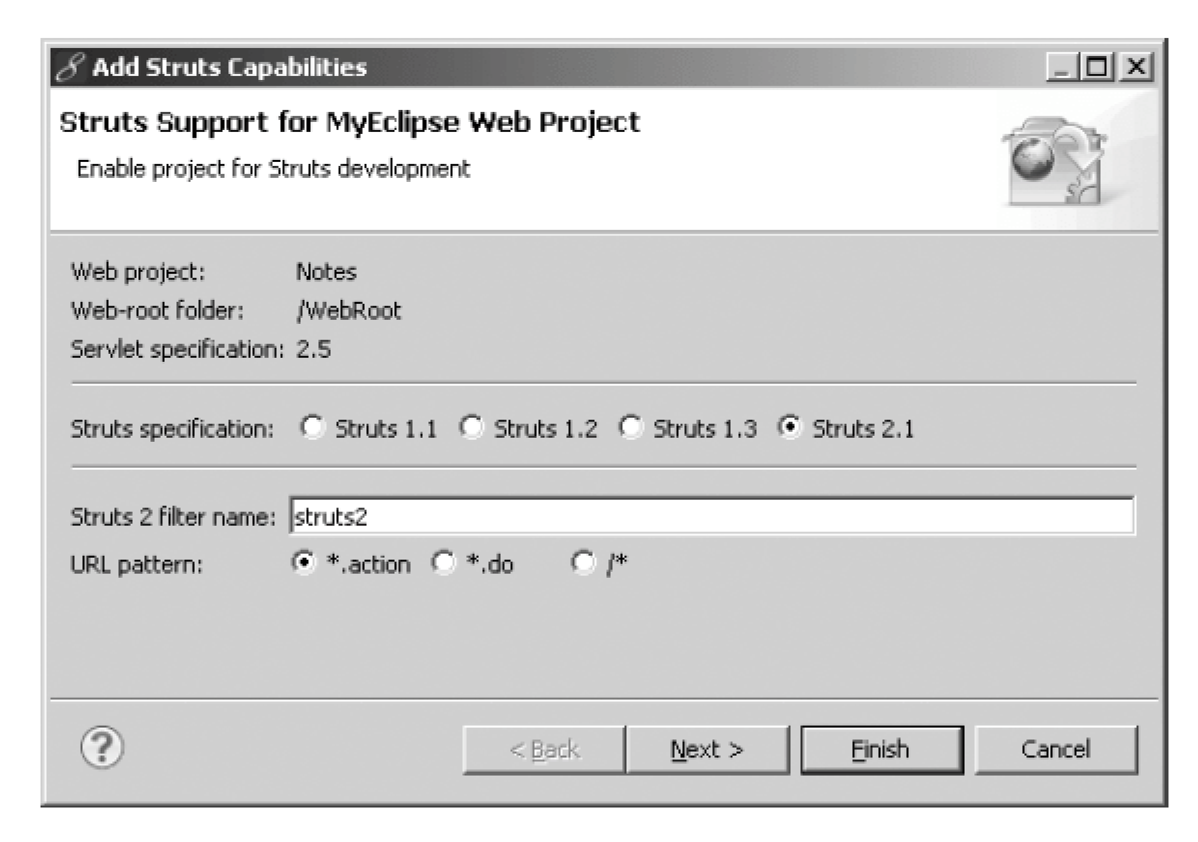


图 4-3 Add Struts Capabilities 向导

在 Add Struts Capabilities 向导中，可以通过 Struts specification 选项组选择需要添加的 Struts 2 版本。由于开发时使用的是 Struts 2 框架，因此选择 Struts 2.1 选项。利用 URL pattern 选项组选择需要交由 Struts 2 处理的服务请求的类型，其中各选项含义如下。

(1) *.action：仅将所有扩展名为.action 的 URL 请求交给 Struts 2 处理。

(2) *.do：仅将所有扩展名为.do 的 URL 请求交给 Struts 2 处理，这是模拟 Struts 1.x 框架中的请求 URL。

(3) /*：所有服务请求均交给 Struts 2 处理。

单击 Next 按钮，进入下一步，为应用程序添加 Struts 2 支持库和自定义的类库，如图 4-4 所示。默认时，MyEclipse 仅为应用程序添加 Struts 2 的核心库。用户可以根据需要添加 Struts 2 DOJO 等类库。

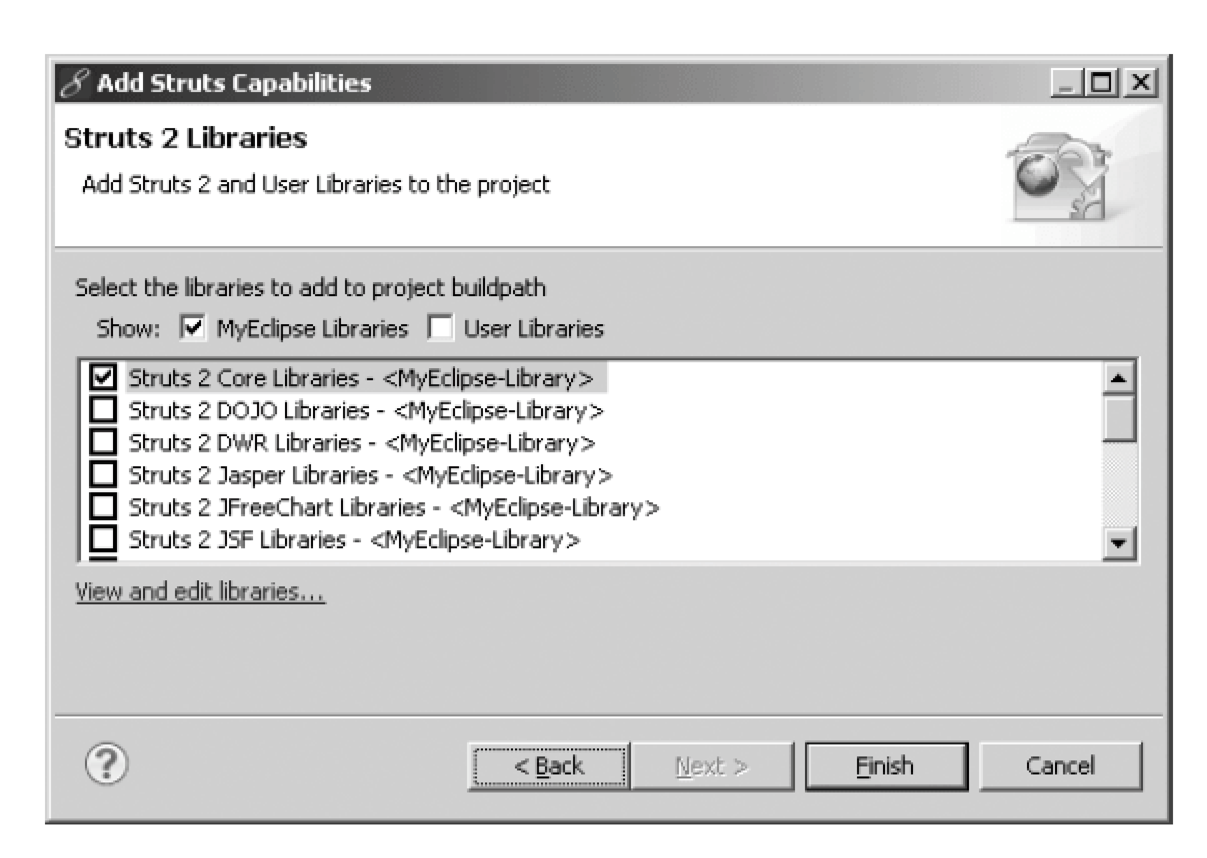


图 4-4 添加 Struts 2 支持库

默认情况下，MyEclipse 将为应用程序添加四十几个 JAR 包，其中有很多包在用户

程序中可能用不到。为此，用户可以选择 Struts 2 Core Libraries，然后单击 View and edit libraries...链接，在打开的 Preferences(filtered)对话框中删除不必要的 JAR 文件。在 Preferences(filtered)对话框中选择 Struts 2 后，可以看到 MyEclipse 为应用程序添加的与所有 Struts 2 有关的 JAR 文件，如图 4-5 所示。仅保留第 4 章中介绍的前 6 个即可。



图 4-5　Preferences(filtered)对话框

注意：利用 MyEclipse 的 Struts 2 支持向导添加的 Struts 2 版本是 Struts-2.1.8.1，不需要 Javassist-x.x.ga.jar。

回到 Add Struts Capabilities 对话框，然后单击 Finish 按钮，完成添加 Struts 2 支持。此时，单击包浏览器 Package Explorer，可以看到上述操作过程的结果。

(1) Struts 2 支持包已经添加到应用程序中。

(2) 在 src 文件夹下添加了一个 struts.xml 配置文件。

(3) 打开 web.xml，可以看到 Struts 2 的核心过滤器配置完成。

最后，需要提醒的是，利用向导添加的 Struts 2 支持包来源于 MyEclipse 安装目录，版本相对比较老，如果读者希望使用相对较新的开发包，则建议手工配置。

同 步 训 练

1. 为留言板程序编写 Action 类。
2. 编写站内短信系统的 Action 类。

Chapter 5 第5章 深入 Struts 2

5.1 Struts 2的工作原理

在体验完利用 Struts 2 开发 Web 应用程序的过程之后，我们来看一下 Struts 2 的工作原理。

Struts 2 框架是在 Struts 1 和 WebWork 基础上发展而来，在实际的 Web 应用开发过程中，主要用于解决与表示层的相关问题。Struts 2 的核心架构是基于 MVC 设计模式设计，Model 由程序员编写的一系列 Action 以及一些用于实现业务逻辑方法或与底层数据库交互的实体类组成；View 通常是指 JSP 页面或者其他视图显示技术，如 FreeMarker 或者 Velocity 等；Control 是 Struts 2 框架提供的 FilterDispatcher(现在已由 StrutsPrepareAndExecuteFilter 取代)核心控制器，负责根据客户请求调用相应的模型组件 Action，再由 Action 调用相应的业务逻辑组件来完成处理。

图 5-1 是 Struts 2 官方文档中给出的 Struts 2 体系结构图。

Struts 2 体系结构中包括的部件比较多，主要有以下几项。

(1) ActionMapper：用于提供 HTTP 请求与 Action 执行之间的映射。当请求到达时，ActionMapper 根据 web. xml 中核心控制器的配置判断是否应该由 Struts 2 框架处理该请求。如果应该处理，则 ActionMapper 返回一个对象来描述请求对应的 ActionInvocation 的信息。

(2) FilterDispatcher：已经由 StrutsPrepareAndExecuteFilter 替代，是 Struts 2 的调度中心，在获得 ActionMapper 的处理通知后，负责停止过滤器链上还没有执行的过滤器，执行 Action 处理。

(3) ActionProxy：充当了 Action 和 xwork 之间的代理，负责初始化 Action 运行所需的所有参数配置等。ActionProxy 持有 ActionInvocation 对象。

(4) ConfigurationManager：提供对配置文件 struts. xml 的访问。该对象在 Web 应用程序开始运行时，将 struts. xml 读入内存。

(5) struts. xml：Struts 2 的应用配置文件，负责配置请求与 Action 之间的映射，以及配置 Action 结果码对应的 Result 视图等。

(6) ActionInvocation：负责调度 Interceptor、Action 和 Result。

(7) Interceptor(拦截器)：拦截器类似于 Servlet 中的过滤器，提供了在 Action 运行之前或 Result 运行之后执行一些功能代码的机会。

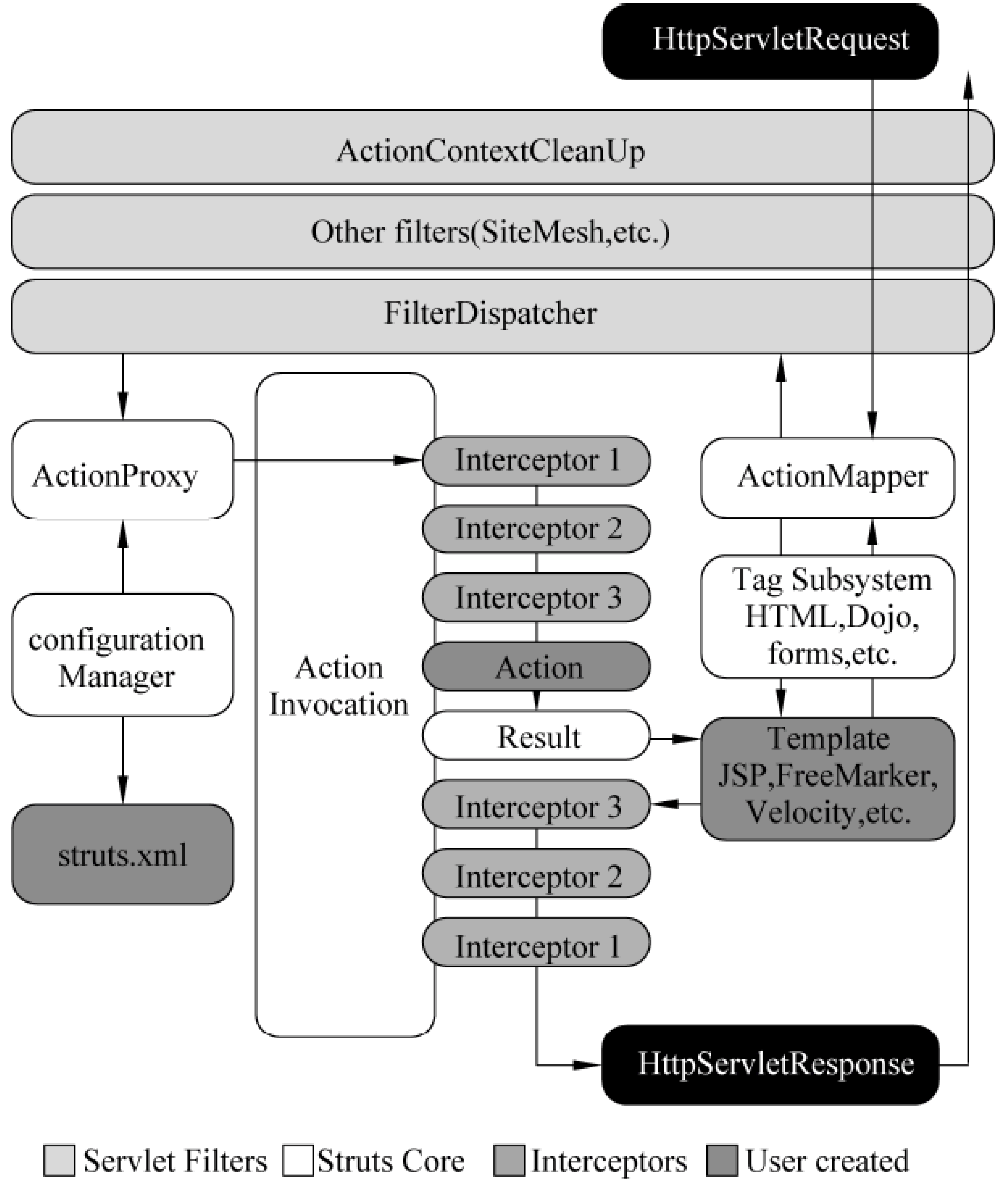


图 5-1　Struts 2 的体系结构

(8) Action：负责处理用户请求，调用业务逻辑组件。

(9) Result：负责映射 Action 执行后需要呈献给客户端的结果视图，可以是一个 JSP 页面或是某个 Action。

(10) Templates：各种视图类型的页面模板。Struts 2 集成了三种模板：JSP、FreeMarker 和 Velocity。

(11) Tag Subsystem(标签库)：Struts 2 的标签库。

(12) ActionContextCleanUp：在集成 SiteMesh 或者其他过滤器时，需要在 Web. Xml 中配置这个部件，推迟核心控制器对 ActionContext 的清空，以便集成的过滤器能够访问到值栈中的数据。

下面结合 4.2 节的例子来研究 Struts 2 框架的请求处理流程。

(1) 当 Servlet 容器接收到客户请求 http://127.0.0.1:8080/login.action 后，首先将请求交给 FilterDispatcher 或 StrutsPrepareAndExecuteFilter 的 doFilter()方法。

(2) 核心控制器的 doFilter()方法询问 ActionMapper 是否应该由 Struts 2 框架处理 login.action。

(3) ActionMapper 告诉核心控制器，需要处理 login.action，核心控制器会停止过滤器链以后的部分(本例中没有)。

(4) 核心控制器调用 Dispatcher 类的 serviceAction()方法。

(5) Dispatcher 调用 ActionProxy 的 execute()方法。

(6) ActionProxy 对象利用从核心控制器中获得的请求的 URL 向 ConfigurationManager 查询 Login.Action 对应的实现类。

(7) ActionProxy 创建 ActionInvocation 对象,设置它的执行上下文 ActionContext,并调用其 invoke()方法。从这一步也可以看出,第 4 章所说的 ActionContext 是线程安全的,原因就在于每个请求的 ActionInvocation 对象是不同的,每个 ActionInvocation 对象有自己的 ActionContext,所以不同的 Action 之间是不会共享 ActionContext 的。

(8) ActionInvocation 对象的 invoke()方法查找所有未执行的拦截器,例如 params 拦截器将请求中的参数 userName 和 password 装配到 LoginAction 实例的同名属性上等。

(9) 在所有拦截器执行完毕后,调用 LoginAction 实例的 execute()方法(在其他应用程序中可能是其他方法,具体取决于 Action 的配置)。

(10) ActionInvocation 对象按照拦截器引用顺序的倒序依次执行各个拦截器的后置部分。

(11) ActionInvocation 根据 LoginAction 实例执行返回的结果码(假设为 success),查找 struts.xml 中对应的 result 映射。执行 Result 的 execute()方法从 ServletActionContext 中获得 HttpServletResponse 对象,利用该对象将结果页面/welcome.jsp 返回给浏览器。

5.2 Struts 2 的配置文件

5.2.1 Struts 2 的配置文件介绍

每个 Struts 2 应用程序都要用到几个配置文件,其中 web.xml、struts.xml 和 struts.properties 文件是最基本的。web.xml 用于初始化 Web 应用程序的配置信息,例如加载 Struts 2 的核心控制器等。struts.xml 文件主要负责管理 Web 应用中的 Action 映射,以及该 Action 每种结果代码对应的 result 定义等。struts.properties 文件主要用于配置常量。

除了上述 3 个配置文件外,Struts 2 的配置文件还包括 struts-default.xml 和 struts-plugin.xml。struts-default.xml 是 Struts 2 的基础配置文件,定义了内置的结果类型、拦截器和拦截器栈,由框架自动加载。struts-plugin.xml 用于为 Struts 2 添加插件(plugin)。通过插件,可以扩展、替换 Struts 2 中的某些功能点,也可以加入自己的实现类,使得 Struts 2 具备新的功能。插件的方式也使得任何功能扩展都与 Struts 2 的主体程序保持独立性。

5.2.2 struts.xml 的结构

struts.xml 文件是一个以 struts 作为根元素的 XML 文件,Struts 2 应用程序的很多设置都需要在该文件中完成。

在 struts.xml 可以完成 constant(常量)、package(包)、namespace(命名空间)、

include(包含)、interceptor(拦截器)、action、result、exception(异常)和 bean 配置。在实际的开发中,很少在 strust. xml 配置 bean,感兴趣的同学可以自己查找相关资料,本书不再讨论。

5.2.3　constant(常量)配置

org. apache. struts2 包中包含了一个 default. properties 属性文件,该文件定义了一系列常量,这些常量给出了 Struts 2 应用运行时的默认配置,程序员可以通过修改这些常量来改变相应的配置。

常量的修改可以在 web. xml、struts. xml 和 struts. properties 文件中完成。通常,Struts 2 框架按照如下搜索顺序加载 Struts 2 常量。

(1) struts-default. xml

(2) struts-plugin. xml

(3) struts. xml

(4) struts. properties

(5) web. xml

也就是说,如果在多个文件当中配置了同一个 Struts 2 常量,则后一个文件中配置的常量会覆盖前面文件中配置的常量。

下面给出在 Web 应用开发过程中常用的一些常量,读者可以通过 default. properties 文件查看所有的常量定义。

(1) struts. i18n. encoding。用于指定默认的编码方案,默认值为 UTF-8。

(2) struts. custom. i18n. resources。用于指定要加载的国际化资源包的基名。如果包含多个资源包,则需要用逗号分隔。

(3) struts. locale。用于指定 Web 应用的默认 Locale。该属性未设置默认值。

(4) struts. devMode。用于指定是否处于开发模式,默认值为 false。通常,在应用程序开发过程中需要将该常量设置为 true。此时,在调试过程中,Struts 2 框架能够提供更为友好的错误报告方式,提醒程序员更多的错误信息。

(5) struts. enable. DynamicMethodInvocation。用于设置是否支持动态方法调用,默认值为 true。出于安全考虑,通常要将该常量设置为 false,即禁止动态方法调用。

1. 在 struts.properties 中配置常量

struts. properties 文件和标准的 Properties 文件一样,包含了一系列 key-value 对象,每个 key 就是一个 Struts 2 属性,该 key 对应的 value 就是一个 Struts 2 属性值。程序员可以通过在 struts. properties 文件中修改 default. properties 中定义的属性,来满足应用的需求。

在利用 MyEclipse 开发 Struts 2 应用时,struts. properties 应该放在 src 目录下,Web 应用发布时会自动发布到/WEB-INF/classes 目录下。代码 5-1 演示了如何在 struts. properties 中配置常量。

代码 5-1　在 struts. properties 中配置常量

```
struts.devMode=true
struts.enable.DynamicMethodInvocation=false
```

2 在 struts.xml 中配置常量

也可以在 struts. xml 中利用以下语句配置 Struts 2 常量。

```
<constant name="" value=""></constant>
```

代码 5-2 演示了在 struts. xml 配置 Struts 2 常量的方法，其效果与代码 5-1 等价。

代码 5-2　在 struts. xml 配置常量

```
<struts>
    <constant name="struts.devMode" value="true"></constant>
    <constant name="struts.enable.DynamicMethodInvocation" value="false">
    </constant>
    ...
</struts>
```

常量的配置通常都是在 struts. xml 文件中进行，一般不需要在 struts. properties 文件中配置 Struts 2 常量。Struts 2 框架之所以要保留 struts. properties 文件，主要是为了保持与 WebWork 的向后兼容性。

5.2.4　package（包）配置

与 Java 语言类似，Struts 2 框架也使用包来管理 Action 和拦截器等。每个包就是一些 Action、拦截器、result-types 的集合。使用 package 可以对 Struts 2 的配置内容实现模块化管理，每个 package 中存放的都是逻辑上相关的组件。

Struts 2 中的 package 类似于 Java 中的类，允许从一个 package 的基础上扩展生成另一个 package。

package 元素通常需要对在 struts-default. xml 文件里定义的 struts-default 包进行扩展，从而使扩展之后的包可以使用在 struts-default. xml 中定义的结果类型、拦截器和拦截器栈等。

package 元素的形式如下：

```
<package name="" extends="" abstract="" namespace="">
    ...
</package>
```

其中，每个 package 元素必须提供一个 name 属性值，用于标识包的名字。name 属性的取值必须是唯一的，只有在该 package 被其他 package 扩展时引用。abstract 属性允许将该 package 设置为抽象的。抽象的 package 不能定义 Action。namespace 属性用于定义 package 的命名空间，将在下一节讨论。

注意：子包中配置的 Action 和拦截器引用将覆盖父包中配置的 Action 和拦截器引

用。有关拦截器的配置，将在后续章节中讨论。

5.2.5　namespace（命名空间）配置

package 元素提供了一个 namespace 属性。通过 namespace，程序员可以将所有的 Action 配置划分为一个个逻辑单元，每个单元都有它自己的标识前缀，从而避免 Action 命名的冲突。在访问某个 Action 时，通常使用形如 namespace/actionname 的 URI。

package 元素的默认命名空间为""。Struts 2 框架在查找 Action 时，如果其他所有的 namespace 中都找不到的时候，就会到这个 namespace 中寻找。

Struts 2 支持根 namespace"/"。对于包含在根 namespace 中的 Action，在访问时，通常使用形如 actionname 的 URI。

如果一个 Action 没有指定任何命名空间，如直接是 moo.action，它则会去根命名空间寻找。如果一个 Action 在指定的命名空间没被发现，Struts 2 就会去默认命名空间寻找。

代码 5-3 给出一个 namespace 示例，假设 Web 应用名为 bbs。

代码 5-3　namespace 举例

```
<!-- default 包在默认的名称空间中 -->
<package name="default" extends="struts-default">
    <action name="login" class="bbs.action.LoginAction">
        <result name="success">/index.jsp</result>
    </action>
    <action name="post" class="bbs.action.PostAction"></action>
</package>

<!-- teacher 包在 / 名称空间中 -->
<package name="teacher" namespace="/" extends="struts-default">
    <action name="login" class="bbs.action.LoginAction3">
        <result name="success">/teacher/index.jsp</result>
    </action>
    <action name="post" class="bbs.action.PostAction2"></action>
</package>

<!-- manager 包在根名称空间中 -->
<package name="manager" namespace="/manager" extends="struts-default">
    <action name="managerLogin" class="bbs.action.LoginAction2">
        <result name="success">/manager/index.jsp</result>
    </action>
</package>
```

(1) 当请求 http://127.0.0.1:8080/bbs/login.action 时，Struts 2 首先在根 namespace（"/"）中查找。如果找不到，则到默认的 namespace 中查找。由于根 namespace 中存在 login.action，所以执行 bbs.action.LoginAction3 的实例。

(2) 当请求 http://127.0.0.1:8080/bbs/manager/login.action 时，Struts 2 首先

在/manager namespace("/")中查找。由于该 namespace 中不存在 login. action,所以到默认的 namespace 中查找,也就是执行 bbs. action. LoginAction 的实例。

(3) 当请求如下三个请求时,前两个请求执行根 namespace 中的 post. action,后一个请求执行默认 namespace 中的 post. action

① http://127.0.0.1:8080/bbs/post. action

② http://127.0.0.1:8080/bbs/student/post. action

③ http://127.0.0.1:8080/bbs/manager/post. action

这里需要注意的是第二个请求。在请求 post. action 时使用了/student/post. action 的格式,而实际上根本没有名为 student 的 namespace 存在,Web 应用程序没有出错,也没有直接去查找默认的 namespace,而是首先查找根 namespace 中是否存在 post. action。

5.2.6 include(包含)配置

package 和 action 等的默认配置文件为 struts. xml,但是当一个应用的 package、action、interceptors 等配置比较多时,如果都放到一个 struts. xml 文件中,则维护起来比较困难。为此,把 struts. xml 文件分成多个配置文件,然后在 struts. xml 文件中使用<include>元素包含这些配置文件,使得配置文件的结构更加清晰,维护起来更加容易。

例如在一个电子商务应用中,可以把用户配置、商品配置、订单配置分别放在 3 个配置文件 user. xml、goods. xml 和 order. xml 中,然后在 struts. xml 中将这 3 个配置文件引入,如代码 5-4 所示。

代码 5-4　带有 include 元素的 struts. xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>

        <include file="user.xml"></include>
        <include file="goods.xml"></include>
        <include file="order.xml"></include>
</struts>
```

代码 5-5 给出了被包含文件 user. xml 的片段。

代码 5-5　user. xml 的代码片段

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>

    <package name="user" extends="struts-default">
```

```
        <action name="login" class="org.shops.action.User.Login">
            <result name="success" type="redirect">/index.jsp</result>
        </action>
        <action name="register" class="org.shops.action.User" method="reg">
                <result name="success" type="redirect">/index.jsp</result>
        </action>
        ...
        </package>

    </struts>
```

被包含的文件本身也是一个完整的配置文件，必须遵守 struts-2.3.dtd 的定义。在利用 myEclipse 开发 Web 应用时，被包含的文件可以存放在 src 目录下，也可以存放在某个 Java 包下。例如，当上述 user.xml 存放在 org/shops/action/User 下时，在 struts.xml 中应该写成如下形式：

```
<include file="org/shops/action/User/user.xml"></include>
```

5.3　配置 Action

5.3.1　使用 method 属性

Struts 2 使用包来管理 Action 信息，需要在 struts.xml 文件中使用<package>元素的子元素<action>来配置 Action 信息。

当客户端请求一个 Action 时，Struts 框架根据 struts.xml 中的配置，自动将请求交给 action 元素的 name 属性所标识的 Action 类的实例去处理。默认时，调用该实例的 execute()方法。

在实际的应用开发中，为了减少 Action 类，不会在一个 Action 类中只包含一个功能，而是把几个相关的功能放在同一个 Action 类中。例如在电子商务应用中，围绕订单可能包括向订单中添加商品、从订单中删除某一个商品、提交订单、查询订单等功能。为减少 Action 类的使用，可以将这些功能放在同一个 Action 类中，如代码 5-6 给出的 OrderAction.java 类的代码片段所示。

代码 5-6　OrderAction.java 片段

```
public class OrderAction extends ActionSupport {

    public String execute() {            //向订单中添加商品
        ...
        return SUCCESS;
    }

    public String submit() {             //提交订单
```

```
        ...
        return SUCCESS;
    }

    public String view(){              //查看订单
        ...
        return SUCCESS;
    }

}
```

当一个 Action 类中包含多个功能时，在配置 Action 映射时，需要使用<action>元素的 method 属性来指定 action 调用的方法。

代码 5-7 中配置了 3 个 Action，分别用于访问 OrderAction 类的 execute()、submit()和 view()方法。

代码 5-7 在 struts.xml 中为同一个 Action 类配置不同的别名

```
<package name="orders" namespace="/orders" extends="struts-default">
    <!--访问 add.action 时,将调用 OrderAction 的 execute()方法 -->
        <action name="add" class="org.shops.action.OrderAction">
            <result name="success" type="redirect">/orders.jsp</result>
        </action>

    <!--访问 submit.action 时,将调用 OrderAction 的 submit()方法 -->
        <action name="submit" class="org.shops.action.OrderAction"
         method="submit">
            <result name="success" type="redirect">/orders.jsp</result>
        </action>

    <!--访问 view.action 时,将调用 OrderAction 的 view()方法 -->
        <action name="view" class="org.shops.action.OrderAction"
         method="view">
            <result name="success" type="redirect">/orders.jsp</result>
        </action>
</package>
```

5.3.2 动态方法调用

在代码 5-7 中，利用 3 个 Action 分别访问 OrderAction 类的 3 个不同方法。实际上，在 Struts 2 中提供了一种被称为动态方法调用的方法。使用这种方法，不需要配置就可以直接调用 Action 中的非 execute 方法。

动态方法调用（Dynamic Method Invocation，DMI）是指使用 actionName!methodName.action 的形式来访问 Action 类中的方法。其中，actionName 为在 struts.xml 中为某个 Action 类的 execute()方法配置的 Action 名字，methodName 为该 Action

类中的非 execute()方法。

例如，对于代码 5-6 中的 OrderAction 类，如果在 struts. xml 中进行如代码 5-8 所示的配置，就可以利用 order. action!del. action 调用 OrderAction 类的 del()方法。

代码 5-8　动态调用类 OrderAction 中的方法时的 Action 配置

```
<action name="order" class="org.shops.action.OrderAction">
    <result name="success" type="redirect">/orders.jsp</result>
</action>
```

利用动态方法调用，使得同一个 Action 类的不同方法共用同一个配置。然而，根据 Struts 2 的官方提示，最好不要使用动态方法调用，因为这会引发安全问题。程序员和网站管理者不会希望用户能够调用没有公开的方法。

如果非要使用动态方法调用，则应首先确保常量 DynamicMethodInvocation 没有被设置为 false。

5.3.3　使用通配符

在一个大型的 Web 应用中，可能会包括几十个甚至是上百个 Action 映射。Struts 2 提供的通配符机制允许把相似的映射关系简化成一个映射关系。

表 5-1 给出了在 Action 配置时可以使用的通配符和特殊记号。

表 5-1　Action 配置时可以使用的通配符和特殊记号

通配符或记号	含　义
*	可以匹配 0 个或多个任意字符，但是不包括斜杠“/”
**	可以匹配 0 个或多个任意字符，包括斜杠“/”在内
\	反斜杠，转义字符
{n}	表示第 n 个通配符所匹配的值。n 为数字，取值范围 0～9，当 n 为 0 时，{n}匹配整个请求 URI

先来看一下代码 5-9 给出的 Action 配置。

代码 5-9　在 Action 配置时使用通配符

```
<action name="order_*" class="org.shops.action.OrderAction" method="{1}">
    <result name="success" type="redirect">/orders_{1}.jsp</result>
</action>
```

在代码 5-9 给出的 Action 配置中使用了通配符“*”，因此该 Action 可以匹配所有以/order_为前缀的 URI，例如/order_add、/order_submit 和/order_remove 等。当客户请求 URI 为 order_add 时，由于通配符“*”匹配了 add，{1}将被替换成 add，因此该请求将调用 org. shops. action. OrderAction 的 add 方法；当 Action 实例返回 SUCCESS 时，结果页面将映射到/orders_add. jsp。

接下来，看一下代码 5-10 给出的 Action 配置。

代码 5-10　在 Action 配置时使用通配符的另一个例子

```
<action name="*_*" class="org.shops.action.{1}Action" method="{2}">
    <result name="success" type="redirect">/{1}_{2}.jsp</result>
</action>
```

在代码 5-9 给出的 Action 配置中使用了 *_* 来为 Action 命名，因此该 Action 可以匹配所有中间带有下画线（“_”），但是不含有“/”的 URI 请求，例如/order_add、/user_register 和/goods_remove 等。当客户请求 URI 为 goods_remove 时，由于第 1 个通配符匹配了 goods，第 2 个通配符匹配了 remove，因此{1}将被替换成 goods，{2}将被替换成 remove，该请求将调用 org. shops. action. goodsAction 的 remove 方法。当 Action 实例返回 SUCCESS 时，结果页面将映射到/goods_remove. jsp。

由于{0}可以匹配整个请求 URI，因此代码 5-10 中的{1}_{2}. jsp 也可以写成{0}. jsp，其效果是完全一样的。

在 Action 配置时，如果使用了通配符，则可能出现多个 Action 映射都能够匹配用户请求的情况。

来看一下代码 5-11 给出的配置文件片段。

代码 5-11　多个 Action 映射匹配同一个用户请求

```
<action name="*" class="org.shops.action.OrderAction">
    <result name="success" type="redirect">/orders.jsp</result>
</action>
<action name="order_*" class="org.shops.action.OrderAction" method="{1}">
    <result name="success" type="redirect">/orders_{1}.jsp</result>
</action>
<action name="order_add" class="org.shops.action.OrderAction" method="add">
    <result name="success" type="redirect">/orders/orders.jsp</result>
</action>
```

当请求/order_add. action 时，尽管 3 个 Action 映射都可以匹配用户请求，但 Struts 2 框架会优先选择第 3 个映射，即名字为 order_add 的 Action 映射。

当请求/order_submit. action 时，Struts 2 是不是优先使用第 2 个 Action 映射来匹配客户请求呢？经过验证发现，Struts 2 选择了第 1 个 Action 映射，也就是名字为“*”的 Action 映射来匹配客户请求。如果将代码 5-11 中前两个 Action 映射交换顺序，再次请求/order_submit. action，会发生什么变化呢？验证表明，此时 Struts 2 选择了名字为“order_*”的映射来匹配客户请求。

总结一下，Action 请求的优先级顺序如下。

（1）不带有通配符的 Action 映射的优先级最高，会优先被选择。

（2）带有“*”通配符的 Action 映射将按照其出现在配置文件中的顺序进行匹配请求。

5.3.4　利用静态参数给 Action 传递值

在前面的例子中，Action 类属性的值通常利用表单的输入或 URL 请求参数进行填

充。除了这两种方式外，Struts 2 还允许在 struts. xml 中配置 Action 时利用静态参数进行填充。

struts. xml 文件中的 action 元素可以包含任意个 param 元素。每个 param 元素将映射到 Action 类的一个属性上。Struts 2 框架提供了一个 Static Parameters 拦截器，该拦截器能够把利用 param 元素定义的静态参数的值传递给 Action 的属性。

在代码 5-12 给出的 struts. xml 片段中，使用了一个利用静态参数给 Action 传递值的例子。

代码 5-12　利用静态参数给 Action 传递值

```
<action name="addNote" class="example.action.NotesAction" method="add">
    <param name="uploadDir">uploadFiles</param>
</action>
```

上述代码为 addNote. action 设置了一个静态参数 uploadDir，每当该 Action 运行时，它的 uploadDir 属性都会自动被设置为 uploadFiles。

5.3.5　默认的 Action

通常情况下，当请求的 Action 不存在时，Web 服务器会给客户端返回一个 HTTP 404 错误。为了提供一个友好的错误处理页面，可以配置一个默认的 Action。在请求的 Action 不存在的情况下，调用默认的 Action。

代码 5-13 给出利用 default-action-ref 元素声明默认 Action 的方法。注意，default-action-ref 元素的声明必须放在一个 package 的所有 Action 元素声明之前。

代码 5-13　配置默认 Action

```
<package name="default" namespace="/" extends="struts-default">
    <default-action-ref name="error"></default-action-ref>
    <action name="error">
        <result type="httpheader">
            <param name="status">404</param>
        </result>
    </action>
</package>
```

5.4　配置 result

5.4.1　result 映射与结果类型

在 Action 类的方法执行完成后，Struts 2 框架需要向客户端输出一个处理结果，该结果可能是一个 JSP 或 HTML 页面，也可能是另一个 Action 调用。在 struts. xml 中，通常利用 result 指定一个 Action 实例的每一个可能的结果码对应的输出。

一个完整的 result 结构如下：

```
<result name="success" type="dispatcher">
    <param name="location">/ThankYou.jsp</param>
</result>
```

其中，result 元素的 name 属性表示 Action 实例的返回结果码，默认值为 success；type 属性表示结果类型，默认值为 dispatcher；子元素 param 表示映射时所需参数，每个 result 元素可以有 0 到多个 param 子元素，参数 location 表示结果视图对应的资源 URL。

利用 result 属性的默认值，可以将上述语句简化成如下形式：

```
<result>
    <param name="location">/ThankYou.jsp</param>
</result>
```

当 result 只有 location 参数时，可以将其缩写成如下形式：

```
<result>/ThankYou.jsp</result>
```

通过前面的学习我们知道，Action 类用于管理应用程序的状态（返回结果码），那么这些状态对应的结果视图就由结果类型来管理。在 struts-default.xml 中预定义了一些常用的结果类型，如表 5-2 所示。

表 5-2　struts-default.xml 中预定义的结果类型

结果类型	说　明
chain	用来处理 Action 链
dispatcher	用来转向页面，通常处理 JSP
freemarker	用来集成 FreeMarker
httpheader	用来控制特殊的 HTTP 行为
redirect	用来重定向到一个 URL
redirectAction	用来重定向到一个 Action
stream	用来向浏览器返回 InputSream 对象，通常用来处理文件下载
velocity	用来集成 Velocity
xslt	用来处理 XML/XLST 模板
plainText	用来显示原始文件内容，例如文件源代码

除了 struts-default.xml 中预定义了的结果类型之外，Struts 2 中的某些插件也定义了各自的结果类型。例如，在 struts2-json-plugin-x.x.x.jar 的 struts-plugin.xml 中定义了一个结果类型 json，用于向浏览器返回 JSON 数据。

5.4.2～5.4.5 小节将讲述 result 的各种常见结果类型的配置方法。

5.4.2 dispatcher 类型

dispatcher 是 Struts 2 默认的结果类型，主要用于处理 JSP 页面跳转。

代码 4-12 在 request 中存放了一个字符串“您好，欢迎登录系统！”，为什么能够在代

码 4-13 给出的 welcome. jsp 页面中利用 property 标签将其显示出来呢？通过查看 Struts 2 源码可以看到，dispatcher 类型的实现类是 org. apache. struts2. dispatcher. ServletDispatcherResult，该类在实现时采用了 Servlet 的 RequestDispatcher 来转发请求，这就意味着，结果视图与最初的客户请求将共享 request 和 response 对象。因此，调用 Action 实例的页面中的表单输入，以及 Action 实例在 request 中存放的数据，能够被 JSP 结果视图访问。

dispatcher 有两个参数，一个是 location，用于指定 Action 执行完成后要跳转到的目标 JSP；另一个是 parse，用于标识 Struts 2 是否对 location 参数中的 ognl 表达式进行解析。parse 参数的默认值为 true。

在 location 参数中，可以包含 OGNL 表达式。例如，在某应用程序中允许不同身份的用户登录，并且在登录成功后为不同身份的用户返回不同的结果页面。假设用同一个 Action 类来处理所有身份的用户请求，那么该 Action 类应该具有代码 5-14 所示的形式。

代码 5-14　处理不同身份用户登录的 Action 类

```
…
public class LoginAction extends ActionSupport{
    private int userType;
    private String folder;
    …
    /* 省略了 getter 和 setter 方法的代码,请读者自行添加 */

    public String execute() {
        …
        if(userType==1)
            folder="manager";
        else
            folder="user";

        return SUCCESS;
    }
}
```

对于代码 5-14 中的 LoginAction 类，在 struts. xml 中可以采用代码 5-15 给出的形式进行配置。

代码 5-15　带有 ognl 表达式的 result 配置

```
<action name="login" class="org.shops.action.LoginAction">
    <result>/${folder}/welcome.jsp</result>
</action>
```

这里的 ${folder} 就是 OGNL 表达式，称其为动态值。${folder} 的值来源于 Action 中的属性 folder，所以在 Action 类中必须保证为 folder 属性定义了 get 方法。result 的 parse 属性就是用于控制是否对这样的表达式进行解析的。由于 parse 的默认值为 true，所以代码 5-15 中对其进行了省略。从上面的例子可以看到，当成功登录的用

户为管理员时，浏览器将跳转到/manager/welcome.jsp 页面；否则，浏览器将跳转到/user/welcome.jsp 页面。

另外，dispatcher 对应的结果视图只能是当前应用程序中的页面，不能试图利用 dispatcher 跳转到外部服务器的某一个页面。结果页面在书写时，如果使用"/"作为路径前缀，则表示该结果页面路径是相对于当前的 Web 应用程序的上下文路径；如果结果页面没有使用"/"作为路径前缀，则表示该结果页面的路径相对于当前执行的 Action 的路径。

例如，对于代码 5-16 给出的配置，当请求 http://127.0.0.1:8080/notes/login.action 时，如果 Action 返回结果代码为 success，将跳转到/notes/manager/welcome.jsp 页面；如果 Action 返回结果代码为 error，将跳转到/notes/error.jsp。

代码 5-16 结果页面的配置路径

```
<package name="aaa" namespace="/manager" extends="struts-default">
    <action name="login" class="example.action.LoginAction">
        <result>welcome.jsp</result>
        <result name="error">/error.jsp</result>
    </action>
</package>
```

5.4.3 redirect 类型

与 dispatcher 类型不同，利用 redirect 结果类型，可以将 JSP 页面、Action 以及外部网址作为结果视图。

我们先做一个测试，修改 4.2 节中给出的 Struts 2 例子。对 strus.xml 中名为 login 的 Action 的 result 结果进行修改，将结果码为 success 的结果类型由默认的 dispatcher 修改为 redirect，即

```
<action name="login" class="example.action.LoginAction">
    <result type="redirect">/welcome.jsp</result>
    <result name="login">/login.jsp</result>
</action>
```

利用浏览器访问 http://localhost:8080/notes/，在输入完和 4.2 节相同的用户名和密码后，浏览器将显示如图 5-2 所示的画面。

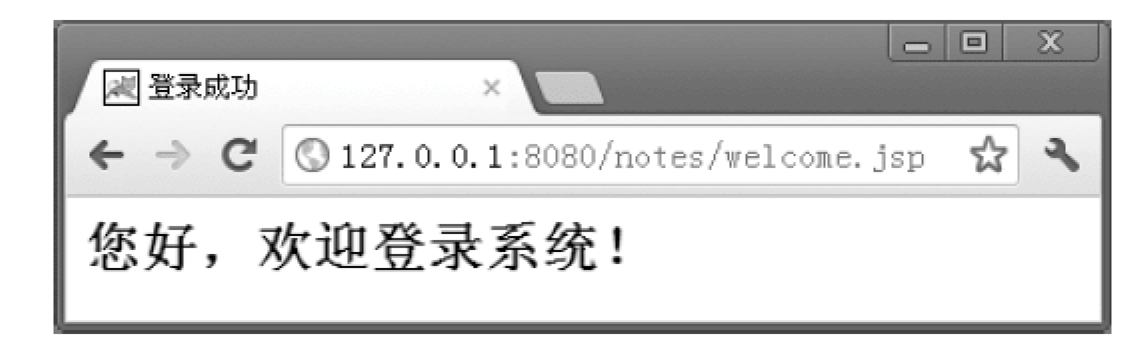


图 5-2 利用 redirect 重定向后的成功页面

仔细比较图 5-2 和图 4-1，会发现两点不同。

（1）图 4-1 中显示出了登录账号 zhangsan；图 5-2 中却没有显示出登录账号。

（2）图 4-1 中显示的 URL 为 http://localhost:8080/notes/login.action，这个 URL 和登录页面（代码 4-3 中给出的 login.jsp）的表单的 Action 的属性值刚好相同；而图 5-2 中显示的 URL 为 http://localhost:8080/notes/welcome.jsp，这个 URL 和 redirect 结果视图是一致的。

之所以会产生这样的不同，原因在于 redirect 结果类型的实现类为 org.apache.struts2.dispatcher.ServletRedirectResult，该类采用 HttpServletResponse 的 sendRedirect 方法将请求重定向到目标资源。也就是说，当采用 redirect 作为结果类型时，Web 服务器向浏览器发送一个重定向操作，该操作将在 HTTP 响应头部封装一个重定向页面，如图 5-3 所示。然后，浏览器向服务器产生一个新的 HTTP 请求，Web 服务器把目标资源返回给浏览器。

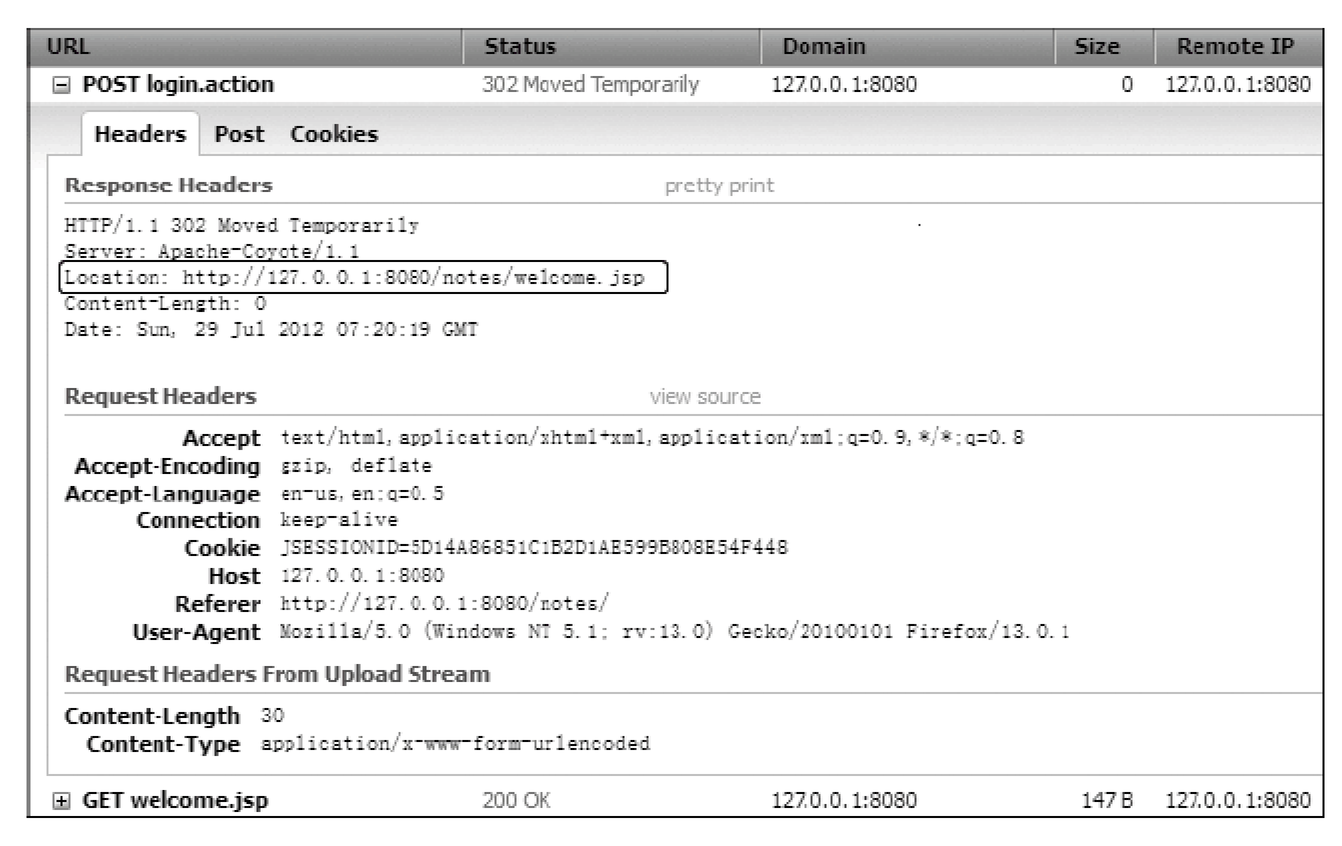


图 5-3　firebug 捕捉的 redirect 结果类型的 HTTP 交互

由于在采用 redirect 作为结果类型时，一次用户交互过程中存在两次 HTTP 请求，Web 服务器对于每一个 HTTP 请求都会创建一个线程进行管理，每一个线程都具有自己的 ActionContext。而 ActionContext 是线程安全的，因此用于处理上一个 HTTP 请求的线程保存的数据无法在新的线程中访问。如果某些数据在目标资源中必须访问，则可以采用两种方式来处理，一种方式是利用 session 传递数据；另一种方式是利用请求参数传递数据。由于 session 中能够保存的数据有限，一般采用第二种方式。对于上述例子，可以按照代码 5-17 配置 Action 的结果映射。

代码 5-17　为结果视图配置请求参数

```
<package name="default" namespace="/" extends="struts-default">
    <!--第一种配置请求参数的方法 -->
        <action name="login" class="example.action.LoginAction">
            <result type="redirect">/welcome.jsp?userName=${userName}
            </result>
        </action>
    <!--第二种配置请求参数的方法 -->
    <!--
```

```
            <action name="login" class="example.action.LoginAction">
                <result type="redirect">
                    <param name="location">/welcome.jsp</param>
                    <param name="userName">${userName}</param>
                </result>
            </action>
        -->
    </package>
```

在重定向后，浏览器显示的 URL 地址为 http://127.0.0.1:8080/notes/welcome.jsp?userName=zhangsan。此时，可以在 welcome.jsp 中利用下述语句获取 userName 参数的值。

```
<s:property value="#parameters.userName[0]"/>
```

在采用第一种方式向目标资源传递结果时，如果重定向后的 URL 中具有两个以上的请求参数，例如 http://127.0.0.1:8080/notes/welcome.jsp?userName=zhangsan&id=1，必须将其中的“&”修改为转义字符“&”，即

```
<result type="redirect">/welcome.jsp?userName=${userName} &amp; id=${id}
</result>
```

redirect 结果类型具有三个参数：location、parse 和 anchor。前两个和 dispatcher 结果类型中的含义一样。参数 anchor 用于指定重定向到目标资源中的锚。

下面对结果类型 redirect 和 dispatcher 进行简单的比较。

(1) URL 不同，采用 dispatcher 作为结果类型，刷新浏览器时，会重新发送 Action 执行请求，会导致 Action 重新执行，Struts 2 框架会重新映射结果视图；采用 redirect 作为结果类型，刷新浏览器时，不会重新发送 Action 执行请求，Action 不会再次执行，浏览器只是把 redirect 的结果视图重新发送给客户端而已。因此，redirect 结果类型将会把用户请求重新定向到另一个资源，而不是把控制权交给该资源；而 dispatcher 结果类型会将整个控制器交给目标资源。

这一点在某些场合很有用，例如在用户注册时，为防止用户单击浏览器的“刷新”按钮，造成两次提交，可以采用 redirect 方式来解决。

(2) 采用 redirect 方式，目标资源不能访问 Action 实例、Action 错误及字段错误；而 dispatcher 能够轻松访问这些数据。

(3) redirect 方式可以将用户重定向到一个外部的资源。在 Action 实例返回状态后，如果只是需要将用户定向到一个内部的资源，并且无须担心用户执行刷新操作时，则采用 dispatcher 方式要更好一些，因为这么做响应速度更快。重定向操作使得浏览器不得不发送一个新的 HTTP 请求。

5.4.4 redirectAction 类型

redirectAction 结果类型与 redirect 结果类型的行为很相似，但 redirectAction 类型不能将目标结果定向到一个网页，只能重定向到另一个 Action 实例。

因为 redirectAction 结果类型的重定向操作也会引发浏览器发送第二个 HTTP 请求，因此第二个 Action 实例无法共享第一个 Action 实例的 ActionContext。如果需要共享数据，则可以采用与 redirect 结果类型相同的方法处理。

redirectAction 结果类型的两个主要参数是 actionName 和 namespace。默认参数 actionName 用于指示需要重定向的目标 Action，namespace 用于指示目标 Action 的命名空间，缺省时表示目标 Action 和当前 Action 位于同一命名中。

代码 5-18 给出了利用 redirectAction 结果类型进行重定位的几种情况。

代码 5-18　利用 redirectAction 配置 result 映射

```
<struts>
    <package name="default" namespace="/user" extends="struts-default">
        <!--重定向到统一命名空间的 list.action -->
        <action name="login" class="bbs.action.LoginAction">
                <result type="redirectAction">list</result>
        </action>

        <action name="list" class="bbs.action.PostAction" >
                <result>/postList.jsp</result>
        </action>
    </package>

    <package name="default" namespace="/manager" extends="struts-default">
        <!--重定向到另一命名空间的第一种方法 -->
        <action name="bulletin" class="bbs.action.BulletinAction" >
                <result >/user/list</result>
        </action>

        <!--重定向到另一命名空间的第二种方法 -->
        <action name="bulletin2" class="bbs.action.BulletinAction">
                <result type="redirectAction">
                    <param name="actionName">list</param>
                    <param name="namespace">/user</param>
                </result>
        </action>
    </package>
</struts>
```

5.4.5　chain 类型

chain 结果类型的用途是构成一个 Action 链，前一个 Action 将控制权转交给下一个 Action，后一个 Action 可以共享前一个 Action 的状态。

chain 结果类型的实现类是 com.opensymphony.xwork2.ActionChainResult，它有三个主要参数：actionName、namespace 和 method。前两个参数的用法与 redirectAction 结果类型中的参数一样，参数 method 用于指示执行目标 Action 中的哪个方法。

代码 5-19 是在 Struts 2 官方文档的基础上稍加修改得到的，演示了 chain 结果类型的用法。

代码 5-19 用 chain 结果类型配置 result 映射

```
<package name="public" extends="struts-default">
    <!--利用默认参数将 creatAccount 链向 login -->
    <action name="createAccount" class="...">
        <result type="chain">login</result>
    </action>

    <action name="login" class="...">
        <!--链向另一个命名空间的 dashboard.action 的 revise()方法 -->
        <result type="chain">
            <param name="actionName">dashboard</param>
            <param name="namespace">/secure</param>
            <param name="method">revise</param>
        </result>
    </action>
</package>

<package name="secure" extends="struts-default" namespace="/secure">
    <action name="dashboard" class="...">
        <result>dashboard.jsp</result>
    </action>
</package>
```

注意：

（1）使用 chain 结果类型配置 result 时，目标 Action 不能有扩展名(.action)；

（2）使用 chain 结果类型配置 result 时，不能试图利用请求参数在 Action 间传递数据，也就是不能出现类似于如下的配置：

```
<result type="chain">login.action?userName="zhangsan"</result>
```

这种配置方式是不行的，因为这里要求配置的是要链接的 Action 的 name，不能传递参数。那么，要传递参数怎么办呢？可以利用 ActionContext 或是 ServletActionContext，将需要传递的数据保存在 request 中。

5.4.6 plainText 类型

plaintext 结果类型用于显示文件的源码。HTTP 默认的服务器响应类型 ContentType 为 text/html。因此当发出如下请求时，浏览器将按照 HTML 语法规则对 welcome.jsp 进行解析，显示出来的是 welcome.jsp 的制作效果，而不是源文件。

```
http://127.0.0.1:8080/welcome.jsp
```

但是某些时候，可能需要在浏览器中向用户显示某个页面的源代码是什么样的。为了显示源码，可以利用 Struts 2 框架提供的 plainText 结果类型来配置 result 结果映射。

plainText 结果类型的实现类是 org. apache. struts2. dispatcher. PlainTextResult，它有两个参数 location 和 charSet。默认参数 location 给出需要显示源代码的页面的路径，可选参数 charSet 用于设置服务器响应的字符编码。

代码 5-20 演示了 plainText 结果类型的用法。

代码 5-20　用 plainText 结果类型配置 result 映射

```
<action name="sourceCode">
    <result type="plainText">
        <param name="location">/welcome.jsp</param>
        <param name="charSet">utf-8</param>
    </result>
</action>
```

在这个例子中，通过访问 sourceCode. action 可以获得 welcome. jsp 的源码。参数 charSet 的值为 utf-8，因此服务器在响应 sourceCode. action 时，服务器响应类型将被设置为 text/plain；charset＝gbk。注意，这里的 charSet 一定要和 welcome. jsp 页面的 charSet 保持一致，否则可能引发中文乱码问题。

5.4.7　全局 result

前面讲到的 result 元素都是作为 Action 的子元素出现的，这被称为局部 result，只能作为本 Action 元素的结果视图。当多个 Action 使用同一个 result 时，可以配置全局 result。全局 result 在某些场合比较有用，例如应用程序的每个页面都会判断用户是否登录。如果没有登录，则都要跳转到登录页面。此时，就可以配置全局 result，让所有 Action 共享这个全局的 result，而不用为每个 Action 都配置一个跳转到登录页面的 result。

全局 result 使用 global-results 标签进行配置。全局 result 除了配置的位置不同外，其他格式与局部 result 是完全一样的。

代码 5-21 演示了全局 result 的配置方法。

代码 5-21　配置全局 result

```
<package name="default" extends="struts-default">
    <!-- 全局 result: login 和 error -->
    <global-results>
        <result name="login">/login.jsp</result>
        <result name="error">/error.jsp</result>
    </global-results>

    <action name="login" class="bbs.action.LoginAction">
            <result>/index.jsp</result>
    </action>
    <action name="post" class="bbs.action.PostAction">
        <result>/post.jsp</result>
        <!-- 名为 error 的局部 result -->
```

```
            <result name="error">/error/error.jsp</result>
        </action>
    </package>
```

在这个例子中配置了两个全局result：login和error。当login.action或post.action返回login结果码时，Struts 2将把/login.jsp响应给浏览器。但是，如果Action返回的结果码为error，Struts 2框架会如何处理呢？

在回答上述问题之前，有必要研究在配置了全局result之后，Struts 2框架寻找result的顺序。假设现在Action类返回一个名为error的结果码，Struts 2将按照下面的顺序查找名为error的result映射。

(1) 查找当前Action是否配置了一个名为error的局部result。如果有，就执行这个result；如果没找到，则执行下一步。

(2) 查找当前package中是否配置了一个名为error的全局result。如果有，就执行这个result；如果没找到，则执行下一步。

(3) 递归查找当前包的父包中是否配置了一个名为error的全局result。如果有，就执行这个result；如果没找到，则执行下一步。

(4) 抛出Exception。

由此可见，在局部result和全局result都有能匹配的result映射时，起作用的是局部result。至此，不难看出上例中，当Action返回的结果码为error时，Struts 2框架的处理方式。

5.5 异常映射

不管如何精细编写应用程序和检查代码，总是会有bug出现。在Java中，通常利用try/catch语句块来捕捉异常。异常映射是Struts 2框架提供的一个处理Action类异常的有力工具。利用异常映射，可以为用户提供一个更加友好的界面，而不是一堆错误代码信息。异常映射允许采取声明式异常处理，或是采用手工编写try/catch的方式抛出异常。Action方法抛出的异常能够被自动捕捉，然后经过映射，指向一个预定义好的result。

在struts配置文件中，异常映射可以通过exception-mapping元素完成。该元素具有两个属性：exception和result。exception属性用于指定需要捕捉的异常类型，例如java.lang.Exception等；result属性用于指定一个result结果映射的名字，该结果映射可以来自于当前的Action，也可以来自于global-results声明。

在Action元素中声明的异常映射称为局部异常映射，在global-exception-mappings中声明的异常映射称为全局异常映射。当异常发生时，Struts 2框架的exception拦截器会按照如下顺序去匹配异常映射（对比局部结果和全局结果的查找顺序，可以很容易地理解局部异常映射和全局异常映射的查找顺序）。

(1) 查找抛出异常的Action中是否声明了针对该异常的映射。如果找到，就执行这个exception的映射配置；如果没有，就执行下一步。

（2）查找当前 package 里面的全局异常映射是否声明了针对该异常的映射。如果找到，就执行这个 exception 的映射配置；如果没有，就执行下一步。

（3）递归地查找父包的全局异常映射是否声明了针对该异常的映射。如果找到，就执行这个 exception 的映射配置；如果没有，就执行下一步。

（4）将 exception 抛出给 Struts 2 去处理。

换句话说，在异常映射匹配时，局部异常映射优先于全局异常映射。

代码 5-22 给出了 Struts 2 官方文档中配置异常映射的例子。

代码 5-22　Java 的异常捕捉方式

```
<struts>
    <package name="default">
        ...
        <global-results>
            <result name="login" type="redirect">/Login.action</result>
            <result name="Exception">/Exception.jsp</result>
        </global-results>
        <!--全局异常映射 -->
        <global-exception-mappings>
            <exception-mapping exception="java.sql.SQLException"
                result="SQLException"/>
            <exception-mapping exception="java.lang.Exception"
                result="Exception"/>
        </global-exception-mappings>
            ...
        <action name="DataAccess" class="com.company.DataAccess">
            <!--局部异常映射 -->
            <exception-mapping exception="com.company.SecurityException"
                result="login"/>
            <result name="SQLException"
                type="chain">SQLExceptionAction</result>
            <result>/DataAccess.jsp</result>
        </action>
        ...
    </package>
```

在这个例子中，当发生 java.sql.SQLException 异常时，Struts 2 框架将异常交给/SQLExceptionAction.action 处理；当发生 java.lang.Exception 异常时，跳转到/Exception.jsp 页面；当发生 com.company.SecurityException 异常时，跳到/DataAccess.jsp 页面。

每当一个 exception-mapping 元素捕捉到一个异常，Struts 2 的 exception 拦截器自动向值栈添加以下两个对象。

（1）exception：表示被捕获异常的 exception 对象。

（2）exceptionStack：存储异常信息的堆栈。

可以在错误处理页面使用 Struts 2 的标签来输出异常信息。

```
<s:property value="exception.message"/>
<s:property value="exceptionStack"/>
```

5.6 案例 2：用 Struts 2 改写留言板的数据模型

本节研究利用 Action 类来封装第 3 章中留言板程序 JSP 页面中用到的业务逻辑。

1. 编写 LoginAction 类

LoginAction 类封装了与用户登录有关的业务逻辑，它取代了第 3 章案例 1 中的 doLogin.jsp 和 logout.jsp 文件，内容如代码 5-23 所示。由于第 3 章给出的留言板的 DAO 层已经封装了与数据库有关的操作，因此 LoginAction 类在对登录用户进行身份验证时，可以直接调用相应的 DAO 接口的方法。

代码 5-23 LoginAction 类

```
package notes.action;
import java.util.Map;
import notes.dao.UserDao;
import notes.dao.impl.UserDaoImpl;
import notes.model.User;
import com.opensymphony.xwork2.*;

public class LoginAction extends ActionSupport {
    private String userName;
    private String password;
    /* 省略了 get 和 set 方法 */

    public String execute() throws Exception {
        UserDao userDao=new UserDaoImpl();
        User user=userDao.findUser(userName, password);  /* 调用 DAO 接口的方法 */
        if(null==user){                 /* 未通过用户验证,重新定向到登录页面 */
            this.addActionError("用户名或密码错误!");
            return LOGIN;
        }
        else{                          /* 将登录用户信息保存到 session 中,返回 success 结果码 */
            ActionContext context=ActionContext.getContext();
            Map session=context.getSession();
            session.put("user", user);
            return SUCCESS;
        }
    }

/* 注销用户登录 */
    public String logout(){
        ActionContext context=ActionContext.getContext();
        Map session=context.getSession();
```

```
            session.clear();                /* 清除 session 中的信息 */
            return LOGIN;                   /* 返回 login 结果码,重新定向到登录页面 */
        }
    }
```

代码 5-24 所示的 NotesAction 类封装了与留言有关的业务逻辑,包括获取留言列表,添加留言和获取留言的详细内容等,它取代了第 3 章案例 1 中的 doLogin.jsp 和 index.jsp 中的 Java 脚本。

代码 5-24　NotesAction 类

```
package notes.action;
import java.util.*;
import notes.dao.NotesDao;
import notes.dao.impl.NotesDaoImpl;
import notes.model.*;
import com.opensymphony.xwork2.*;

public class NotesAction extends ActionSupport {
    private Notes note;
    private List<Notes> notes=new ArrayList<Notes>();
    private int noteId;
    /* 省略了 get 和 set 方法 */

    public String list() {                          /* 列出所有留言的信息 */
        NotesDao notesDao=new NotesDaoImpl();
        notes=notesDao.getAllNotes();               /* 调用 DAO 接口的方法 */
        return SUCCESS;
    }

    public String add() {                           /* 添加一条新的留言 */
        if (null==note)     return INPUT;           /* 留言为空,返回 input 结果码 */

        ActionContext context=ActionContext.getContext();
        Map session=context.getSession();
        User user=(User) session.get("user");
        if (null==user)return LOGIN;                /* 用户未登录,返回 login 结果码 */

        note.setUser(user);
        NotesDao notesDao=new NotesDaoImpl();       /* 调用 DAO 接口的方法 */
        notesDao.addNote(note);
        return SUCCESS;
    }

    public String detail() {                        /* 获取某条留言的详情 */
        NotesDao notesDao=new NotesDaoImpl();
        note=notesDao.getNoteById(noteId);          /* 调用 DAO 接口的方法 */
        return SUCCESS;
    }
}
```

2 配置 Action 类

代码 5-25 给出了留言板程序中用到的 Action 和 result 配置。由于多个 Action 都会返回 LOGIN 结果码，故将其定义为全局 result。由于代码中已经给出了详细的注释，这里不再赘述，请读者自行研究每个 Action 和 result 映射。

代码 5-25 留言板的 struts.xml 文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
    <package name="default" namespace="/" extends="struts-default">
        <!--配置全局 result -->
        <global-results>
            <result name="login">WEB-INF/jsp/login.jsp</result>
        </global-results>

        <!--配置 login.action,对应 LoginAction 的 execute 方法,成功后重定向到
            listNotes.action -->
        <action name="login" class="notes.action.LoginAction">
            <result type="redirectAction">listNotes.action</result>
        </action>

        <!--配置 logout.action,对应 LoginAction 的 logout 方法 -->
        <action name="logout" class="notes.action.LoginAction"
            method="logout" />

        <!--配置 listNotes.action,对应 NotesAction 的 list 方法 -->
        <action name="listNotes" class="notes.action.NotesAction"
            method="list">
            <result name="success">WEB-INF/jsp/index.jsp</result>
        </action>

        <!--配置 addNote.action,对应 NotesAction 的 add 方法 -->
        <action name="addNote" class="notes.action.NotesAction"
            method="add">
            <result name="success" type="redirectAction">
                listNotes
            </result>
            <result name="input">WEB-INF/jsp/post.jsp</result>
        </action>

        <!--配置 detail.action,对应 NotesAction 的 detail 方法 -->
        <action name="detail" class="notes.action.NotesAction"
            method="detail">
            <result name="success">WEB-INF/jsp/detail.jsp</result>
        </action>
```

```
    </package>
</struts>
```

同步训练

1. 为留言板程序添加实现注册功能的 Action 类，并在 struts. xml 对其进行配置。

2. 编写站内短信系统的 Action 类，要求相近的功能放到同一个 Action 中实现，并在 struts. xml 对其进行配置。

Chapter 6 第6章 Struts 2 的标签库

6.1 OGNL 表达式

OGNL(Object Graph Navigation Language,对象图导航语言)是一种强大的表达式语言,它通过简单一致的语法,可以任意存取对象的属性或者调用对象的方法,能够遍历整个对象的结构图,实现对象属性类型的转换等功能。Struts 2 采用 OGNL 作为表达式语言,允许程序员利用 OGNL 访问 ActionContext 中的数据。OGNL 的功能比较多,本书只介绍 Struts 2 程序员在开发 Web 应用程序中能够用到的部分。对 OGNL 表达式感兴趣的读者可以访问 http://commons.apache.org/ognl/。

6.1.1 ActionContext 和 Value Stack

每当一个 Action 被调用,Struts 2 就会创建一个 ActionContext。ActionContext 中保存着该 Action 对象和其他运行时的一些数据,例如请求参数和会话等。Value Stack 指的是位于 ActionContext 中,用于保存对象的一个栈,如图 6-1 所示。该栈保存着一些 OGNL 可以存取访问的数据,例如 OGNL 表达式可以访问的数据,Struts 2 标签产生的一些中间数据等。

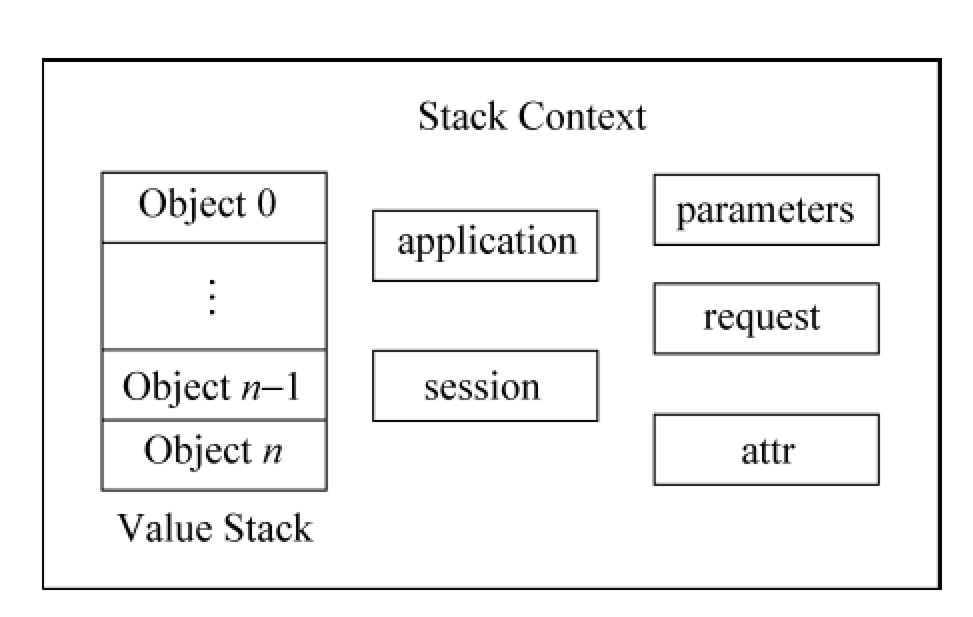


图 6-1　Value Stack

Value Stack 在 Struts 2 中扮演着极其重要的角色。在 Action 处理过程中,拦截器需要访问 Value Stack,JSP 结果页面也需要访问 Value Stack,才能获得 Action 的属性和其他信息。

从图 6-1 可以看出,ActionContext 除了包括 Value Stack 之外,还包含了与 application、session、request、parameters 和 attr 等有关的上下文对象。每一个上下文对象都是一个

Map 对象，使得程序员能够很方便地访问当前应用程序的 ServletContext、当前请求的会话级和请求级参数等。attr 能够依次搜索保存在 request、session 和 application 对象中的数据。

6.1.2　访问 Value Stack 中的元素

可以使用以下几种形式中的某一个访问 Object Stack 里某个对象的属性。

```
对象名.属性名
对象名['属性名']
对象名["属性名"]
```

其中，object 为对象名；property 为对象的属性。

除了直接利用上述形式访问对象的属性外，Struts 2 还允许使用下标的形式访问对象的属性，例如：

```
[0].userName
```

访问的是栈中第一个对象的 userName 属性，也可以写成[0]['userName']或[0]["username"]。

相应地，可以利用[1]. useName 访问栈中的第二个对象的 userName。

在使用[N]. xxx 语法时，要注意位置序号的含义，它并不是表示获取 Value Stack 中索引为 N 的对象，而是从 Value Stack 中的第 N 个位置查找对象 xxx。

如果所要访问的 Value Stack 中的对象本身含有属性，可以使用相同的语法访问这些属性。例如，某个 Action 中有一个 note 对象，note 对象本身又具有一个 title 属性，这时可以用下面的语句访问 title：

```
[0].note.title
```

6.1.3　访问 Stack Context 中的对象

可以通过如下形式访问 ActionContext 中的对象。

```
#对象名.属性名
#对象名['属性名']
#对象名["属性名"]
```

例如，访问请求属性 userName 时，可以使用以下表达式。

```
#request.userName
```

访问会话属性 User 的 userName 属性，可以使用以下表达式。

```
#session.user.userName
```

而表达式

```
#attr.welcome
```

将依次从request、session和application中搜索属性welcome。

6.1.4 访问静态属性和静态方法

Struts 2允许访问保存在Value Stack中的对象的静态属性和静态方法，也允许访问这些对象的public类型的属性和方法。

访问静态属性和静态方法的格式如下：

```
@类名@属性名
@类名@方法名(参数列表)
```

其中，类名为类的全称(含包名)。当省略类名时，默认使用的类是java.lang.Math，例如：

```
@@E                     //访问 java.lang.Math的静态属性 E
@@floor(32.56)          //调用 java.lang.Math的静态方法 floor
```

如果希望访问压入Value Stack中的某个对象的public方法，可以使用如下格式。

```
对象名.方法名(参数列表)
```

6.1.5 访问集合元素

很多情况下，要利用OGNL对一些集合数据进行操作。

1. 数组

可以像访问普通属性那样访问数组元素的属性。例如，Action类中有一个数组属性numbers，并且编写了getNumbers()。

```
String[] numbers={"one","two","three","four"}
```

可以像使用Java中的数组那样去访问numbers，例如：

```
numbers[2]              //返回数组元素中的第 3个元素,此处为字符串"three"
numbers.length          //返回数组元素的长度,此处为 4
```

2. List

在利用Struts 2的标签进行视图设计时，经常需要创建列表。利用OGNL表达式创建列表的方法为：使用花括号将元素括起来，元素之间使用逗号分隔。例如，表达式

```
{"one","two","three","four"}
```

创建了一个长度为4个元素的List对象，元素类型为String。

对于List对象，可以使用下面的表达式进行访问。

```
{"one","two","three","four"}[2]  //返回列表中的第 3个元素,此处为字符串"three"
list.size                        //访问 list中元素的个数,list为一个 List对象
list.isEmpty                     //判断 list是否为空,list为一个 List对象
list.iterator                    //返回用于访问 list的 iterator对象
```

3. Map

利用 OGNL 表达式创建 Map 对象的方法如下：

```
#{key1:value1,key2:value2,…,keyn:valuen}
```

例如，表达式

```
#{"one": "red","two": "blue","three": "white"}
```

创建了一个长度为 3 个元素的 Map 对象，元素类型为 String。

以下是利用 OGNL 访问 Map 对象的例子。

```
#{"one":"red","two":"blue","three":"white"}["two"]   //访问 key="two"的元素
map.three            //返回 Map 对象中 key=three 的 value,此处为"white"
map["two"]           //返回 Map 对象中 key=two 的元素,此处为"blue"
map.size             //判断 Map 是否为空
```

除了上述访问集合对象的方法外，OGNL 还允许利用选择表达式访问集合对象，可以使用的运算符有以下 3 个。

(1) ?：选择符合条件的所有元素。

(2) ^：选择符合条件的第一个元素。

(3) $：选择符合条件的最后一个元素。

例如，users 是一个包含了 User 对象的列表。

```
#users.{?#this.age>30}     //返回所有年龄大于 30 的用户的列表
#users.{^#this.age>30}     //返回年龄大于 30 的第一个用户构成的列表。如果没有
                           //符合条件的元素,返回一个空列表
#users.{$#this.age>30}     //返回年龄大于 30 的最后一个用户构成的列表。如果没
                           //有符合条件的元素,返回一个空列表
```

6.1.6　OGNL 中的三个重要符号

在 Struts 2 中使用 OGNL 时，经常会用到符号“#”、“%”和“$”。

1. “#”符号

“#”符号的主要作用如下。

(1) 用于访问保存在 Stack Context 中的对象。当一个对象 object 保存在 Stack Context 中时，必须使用“#object”的形式去访问。

“#”符号相当于 ActionContext.getContext()。因此，#session.msg 表达式相当于调用 ActionContext.getContext().getSession(). getAttribute("msg") 。

(2) 用于过滤和投影(projecting)集合。例如：

```
#users.{?#this.age>30}
#users.{?#this.age>30}.{age}[0]
```

(3) 用来构造 Map 对象集合。例如：

```
<s:radio list="#{1:'男',2:'女'}" label="性别" name="gender" />
```

中利用“#”符号构造了一个具有两个 Map 对象的集合，并利用该集合充当了 radio 标签的数据源。

2 “%”符号

在为 Struts 2 标签的属性赋值时，除了使用常量外，还可以使用 OGNL 表达式。通常，可以利用“%{”和“}”将 OGNL 表达式括起来。例如，语句

```
<s:property value="%{title}"/>
```

告诉 Struts 2 将 title 的值作为 value 属性的值。

在不发生歧义的情况下，“%”通常可以省略。因此，上面的语句等价于

```
<s:property value="title"/>
```

但是，下面的语句利用 set 标签定义了一个对象 page(值为 10)，并将其保存在 Stack Context 中，然后在 a 标签中将 page 取出，计算下一页的地址。此时就不能省略“%”，即不能将“%{#page-1}”写成“{#page-1}”。

```
<s:set name="page" value="10" id="page"></s:set>
<s:a href="postDetail.action?page=%{#page-1}">上一页</s:a>
```

3 “$”运算符

(1) 在配置文件中使用 OGNL 表达式访问 Action 的属性。例如：

```
<action name="login" class="org.shops.action.LoginAction">
    <result>/${folder}/welcome.jsp</result>
</action>
```

(2) 当在国际化资源文件中构造的消息文本中使用了 OGNL 表达式时，需要将 OGNL 表达式用“${”和“}”括起来。例如：

```
message=欢迎${username}登录系统！
```

详细的使用方法将在后面的国际化部分介绍。

6.2 标签库

6.2.1 使用标签库的好处

在早期的 JSP 设计中，通过在 HTML 标签中嵌入 Java 脚本来生成页面中的动态内容。在一个页面中嵌入过多的脚本片段会大大降低程序的可读性和应用的可维护性。目前，在大型的应用开发中，团队分工比较明确，业务处理逻辑和页面美工设计工作由不同的开发人员承担，页面美工人员往往不懂得 Java 语言，而 Java 程序员对于美工的知识也比较欠缺。因此，原有的 JSP 页面设计方式无法适应实际的开发要求。

JSP 标签库是一种通过 JavaBean 生成基于 XML 的脚本的方法。通过定义标签，可以在简单的标签中封装比较复杂的功能。JSTL 标签库是在 JSP 页面开发中出现最早的

标签库，也是至今仍被很多 Java 程序员热衷的标签库。通过使用 JSTL 标签库，可以获得以下好处。

(1) 标签的应用比较简单，任何人都很容易使用，很容易上手。

(2) 通过标签来表达页面逻辑，可以避免在 JSP 页面中使用 Java 代码，让逻辑与显示分离，提高 JSP 的可维护性。

(3) 通过使用标签，有利于页面美工人员和 Java 程序员进行团队协作开发。

(4) 标签库具有重用性，一旦建立起来，就可以在今后的项目中重复使用。

标签库几乎是每个 MVC 框架的重要组成部分。从 Struts 1 开始，到 Webwork 2、SpringMVC，都有自己定义的一套标签库。Struts 2 也提供了大量的标签，用于简化表示层的设计。

6.2.2　Struts 2 的标签库

Struts 2 中的标签分为通用标签和 UI 标签两大类。其中，通用标签包括数据标签和控制标签两类。数据标签用于显示 Value Stack 中的数据，以及完成国际化等功能；控制标签主要用于实现分支和循环，完成对集合的迭代。UI 标签又分为表单标签、非表单 UI 标签和 Ajax 标签三类。表单标签用于生成 html 元素的标签；非表单 UI 标签用于在 JSP 页面中显示与 Action 有关的消息和错误，显示错误校验信息等；Ajax 标签在实现时调用了 DOJO 等 Ajax 框架。Struts 2.1 之后的版本中不再将 DOJO 作为默认的 JavaScript 框架，这里不介绍这部分标签。

要想在 JSP 页面中使用 Struts 2 的标签，可以在页面开头加上一条 taglib 指令，方法如下：

```
<%@ taglib uri="/struts-tags" prefix="s" %>
```

6.3　数据标签

6.3.1　debug 标签

debug 标签用于帮助程序员进行调试，该标签在页面上生成一个超级链接。点击这个链接，程序员可以查看 ActionContext 和值栈中所有能访问的值。该标签没有任何属性。debug 标签显示的内容如图 6-2 所示。

建议读者在学习后面的各个数据标签时，利用 debug 标签查看每个标签对 Value Stack 和 Stack Context 造成了哪些影响。

6.3.2　property 标签

property 标签的作用是输出 Value Stack 中的数据。property 标签的主要属性如表 6-1 所示。



图 6-2 debug 标签显示的内容

表 6-1 property 标签的主要属性

属性名	是否必需	类 型	说 明
value	否	Object	进行求值的 OGNL 表达式，默认值为 top，此时将返回 Value Stack 最顶端的对象
default	否	String	当 value 为空时的默认值
escape	否	Boolean	是否对输出内容中的 html 特殊字符进行转义。默认值为 true

例如：

```
<s:property value="userName" default="游客" />
```

property 标签经常用于在 JSP 页面中输出 Action 属性的值，或者将 Action 属性的值组装到其他标签中。例如：

```
<img src="images/head/<s:property value='topic.user.head'/>" />
```

中，利用 property 标签动态地获得了用户的头像。

注意 escape 属性的用法。例如，一个留言板程序在用户留言页面可能使用了一个 HTML 在线编辑器，这样，用户可以设置留言内容的字体颜色，甚至添加一个表格。显示留言内容的页面要想正确显示出用户设置的字体颜色以及添加的表格，需要将显示内容的 property 标签的 escape 属性设置为 false；否则，用户只能看到一堆 HTML 代码。

6.3.3 param 标签

param 标签通常用做其他标签的子标签，为其他标签提供运行参数。param 标签的主要属性如表 6-2 所示。

表 6-2 param 标签的主要属性

属性名	是否必需	类 型	说 明
name	否	String	所需设置参数的名字
value	否	Object	所需设置参数的值

param 标签有两种用法。

第一种用法如下：

```
<s:param name="username">zhangsan</s:param>
```

第二种用法如下：

```
<s:param name="username" value="zhangsan"></s:param>
```

注意：上述两种用法不完全等价。用法一的语义是将字符串“zhangsan”赋值给参数 username；用法二的语义是将对象 zhangsan 的值赋值给参数 username，如果对象 zhangsan 不存在，则参数 username 将取空值。如果希望利用用法二给参数 username 赋值为字符串“zhangsan”，则需要使用下述形式。

```
<s:param name="username" value="'zhangsan'"></s:param>
```

再举一个例子：

```
<s:param name="age" value="6+20"></s:param>
```

该语句将参数 age 赋值为 26。

6.3.4　action 标签

action 标签允许程序员直接在 JSP 页面调用一个 Action。在调用 Action 时，可以使用 param 标签向 Action 传递参数。在将 executeResult 属性指定为 true 时，可以将 Action 对应的结果视图也包含到本页面中。action 标签的主要属性如表 6-3 所示。

表 6-3　action 标签的主要属性

属　性　名	是否必需	类　型	说　明
name	是	String	要执行的 Action 的名字，不能写扩展名
namespace	否	String	要执行的 Action 的命名空间
executeResult	否	Boolean	为 true 时，执行 Action 的结果码对应的 result。默认值为 false
ignoreContextParams	否	Boolean	为 true 时，页面的请求参数将传递给 Action。默认值为 false
var	否	String	允许用户根据该属性引用 var

下面举一个关于 action 标签的例子。有个 ActionTagAction 类，如代码 6-1 所示。

代码 6-1　ActionTagAction.java

```
public class ActionTagAction extends ActionSupport {
    private String message;

    //省略 message 属性的 get 和 set 方法

    public String execute() throws Exception{
```

```
            if(null==message)
                message="How are you?";
            ((Map)ActionContext.getContext().get("request"))
                .put("message", message);
            return SUCCESS;
        }

        public String test() throws Exception{

            ((Map)ActionContext.getContext().get("request"))
                .put("message", "现在执行的是 test()");
            return SUCCESS;
        }
    }
```

代码 6-2 给出的页面 actionTag.jsp 利用 action 标签对 ActionTagAction 进行了调用。

代码 6-2 actionTag.jsp 的片段

```
<h3>执行 actionTag,利用 param 向 Action 传递参数,并包含 result 结果</h3>
    <s:action name="actionTag" namespace="/" executeResult="true">
    <s:param name="message">Hello World!</s:param>
    </s:action>
    <s:property value="#attr.message"/>

    <h3>执行 actionTag,禁止将页面请求参数传递给 Action,不包含 result 结果</h3>
    <s:action name="actionTag" namespace="/" ignoreContextParams="true">
    </s:action>
    <s:property value="#attr.message"/>

    <h3>执行 actionTag,不包含 result 结果</h3>
    <s:action name="actionTag" namespace="/" >
    </s:action>
    <s:property value="#attr.message"/>

    <h3>执行 actionTag 中的方法 test,不包含 result 结果</h3>
    <s:action name="actionTag!test" namespace="/" >
    </s:action>
    <s:property value="#attr.message"/>
```

在上述页面中,通过 param 标签向 Action 传递参数,指定 executeResult 属性,决定是否将处理结果的页面包含到当前页面中。通过指定 ignoreContextParams,禁止向 Action 类传递页面的请求参数,通过动态调用执行 Action 的非 execute()方法。程序执行结果如图 6-3 所示。

6.3.5 bean 标签

bean 标签用于实例化一个 JavaBean 对象。在实例化时,可以使用 param 标签为



图 6-3　action 标签的使用

JavaBean 对象的属性赋值。JavaBean 类的定义必须符合规范，也就是说，所有需要使用 param 标签赋值的属性必须定义 set 方法，所有需要在 JSP 页面中取值的属性必须给出 get 方法。表 6-4 给出了常用的 bean 标签的属性。

表 6-4　bean 标签的属性

属性名	是否必需	类 型	说　　明
name	是	String	要实例化的 JavaBean 完整的类名
var	否	String	用于引用该对象

实例化的 JavaBean 将会被压入到 Value Stack 的顶部，在 bean 标签内部可以访问该对象。当 bean 标签结束时，实例化的对象将从 Value Stack 删除。要想在 bean 标签结束之后仍然可以访问实例化的对象，需要用到 var 属性。当指定 var 属性时，JavaBean 在保存到 Value Stack 的同时被保存到 Stack Context 中。在 bean 标签结束时，保存在 Stack Context 中的对象不会被删除，可以通过 var 属性访问，但是需要使用“#”标记。

下面的代码利用 bean 标签实例化了一个 Note 的对象，并利用 var 在 bean 标签结束后访问了对象的 title 属性。

```
<s:bean name="example.Model.Note" var="note">
    <param name="title">请教一个 bean 标签的问题</param>
    <param name="content">能介绍一下 bean 标签的 var 属性吗</param>
</s:bean>
<s:property value="#note.title"/>
```

6.3.6　set 标签

set 标签用于将表达式的值赋给指定范围内的变量。简单地说，set 标签就是定义一个新的变量。表 6-5 给出了 set 标签常用的属性。

表 6-5 set 标签常用的属性

属性名	是否必需	类 型	说 明
name	是	String	变量的名字
value	否	String	设置给变量的值，可以是常量，也可以是 OGNL 表达式。默认时，将 value Stack 栈顶对象赋值给变量
scope	否	String	变量的范围。可选值为 application、request、session、page 和 action。默认值为 action。

一个对象在 ONGL 上的访问层次较深时，例如#request.note.user.userName，就可以使用 set 标签将它定义成一个变量，以保证在多次引用它的时候更方便。set 标签定义的变量将保存到 Stack Context 中。当 scope 取值为 action 时，变量将同时被保存到 request 范围。

代码 6-3 演示了 set 标签的用法。

代码 6-3 set 标签的用法

```
<h3>将 Action 的属性值保存到变量 title 中,并输出</h3>
<s:set name="title" value="note.title"/>
<s:property value="#title"/><br/>

<h3>从 request 请求范围中输出 title</h3>
<s:property value="#request.title"/>

<h3>将常量 hahaha 保存到 session 范围</h3>
<s:set name="test" value="'hahaha'" scope="session"/>
<s:property value="#session.test"/>

<h3>求表达式的值</h3>
<s:set var="page" value="1"/>
当前页:<s:property value="#page"/><br>
<s:set var="page" value="#page+1"/>
下一页:<s:property value="#page"/>
```

注意上例中对于变量的操作。图 6-4 给出了运行时的输出效果。

6.3.7 push 标签

push 标签用于将一个值压入 Value Stack。当一个对象访问层次过深的时候，可以用 push 标签来做访问的简化。和 set 标签不同，push 标签把指定的对象放到 Value Stack 的栈顶，而 set 标签将值放到 Stack Context 中。

push 标签的一个特性是其起始标签把值压入栈中，结束标签将该值弹出。表 6-6 给出了 push 标签的属性。



图 6-4　set 标签运行时的输出效果

表 6-6　push 标签的属性

属性名	是否必需	类 型	说　明
value	是	String	用来指定放到 value stack 栈顶的对象

代码 6-4 演示了 push 标签的用法。

代码 6-4　push 标签的用法

```
<h3>利用 push 标签简化对象的访问 </h3>
<s:bean name="example.Model.Note" var="note">
    <param name="title">请教一个问题</param><br/>
    <param name="content">push 标签和 set 标签有什么区别吗</param>
</s:bean>
<s:push value="#note">
    <s:property value="#title"/><br/>
    <s:property value="#title"/>
</s:push>
```

图 6-5 给出了 push 标签的运行效果。

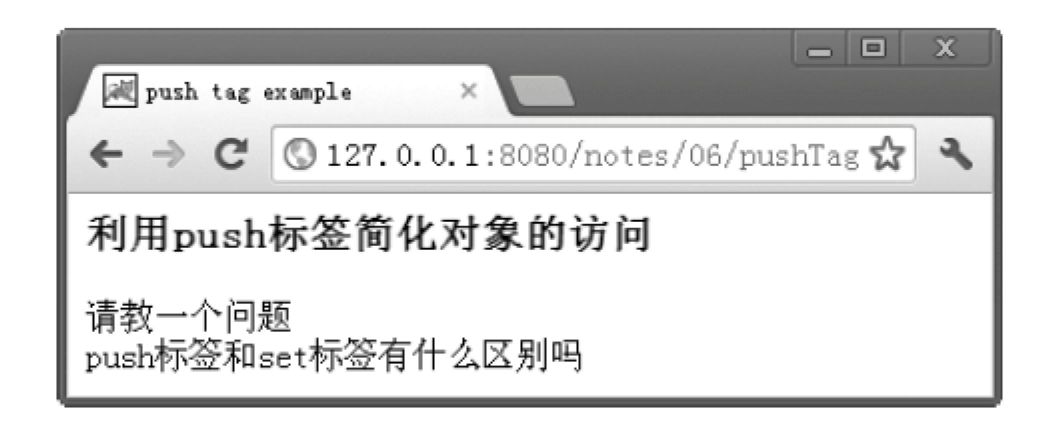


图 6-5　push 标签的运行效果

6.3.8　url 与 a 标签

url 标签用于动态地创建一个 URL。可以利用 param 标签为 URL 提供附加参数。url 标签的属性很多，表 6-7 给出了常用的一些属性。

表 6-7 url 标签的常用属性

属性名	是否必需	类型	说明
action	否	String	指定用生成哪一个 Action 的 URL。省略时，将使用 value 的值作为 URL 地址
value	否	String	指定用生成 URL 的地址值。省略时，将使用 action 的值作为 URL 地址
anchor	否	String	指定 URL 的锚点
includeParams	否	String	指定是否包含请求参数。默认值为 get，可选值为 none、get 和 all
includeContext	否	Boolean	是否将当前上下文路径包含进 URL。默认值为 true
namespace	否	String	指定所用 Action 的命名空间
method	否	String	指定要执行 Action 的方法
scheme	否	String	指定 URL 使用的协议
escapeemp	否	Boolean	是否将“&”转义为“&amp”。默认值为 true
encode	否	Boolean	是否将 URL 编码。默认值为 true
id	否	String	在指定了该值后，URL 不会输出到页面，而是保存在 OGNL 上下文中，以后可以使用 a 标签引用

当 value 和 action 属性同时指定时，优先使用的是 value 属性的值。如果两个属性的值都没有指定，则将使用当前页面的 URL 作为地址值。

如果 includeParams 属性的值为 get，在生成的 URL 中将包含 GET 请求提交的参数；如果值为 all，则在生成的 URL 中会包含 GET 请求和 POST 请求提交的参数；如果值为 none，则在生成的 URL 中不会包含任何请求参数。

a 标签用于创建 HTML 中的标签<a>，它支持 url 标签的所有属性。除此之外，还支持用于指示超级链接地址的属性 href 和绝大多数与标签<a>有关的属性和事件。

代码 6-5 演示了 url 标签和 a 标签的用法。

代码 6-5 url 标签和 a 标签的用法

```
<h3>使用 param 向 url 标签传递参数,使用 action 属性指定地址值</h3>
<s:url action="actionTag">
    <s:param name="message">Hello World!</s:param>
</s:url>

<h3>使用 param 向 url 标签传递参数,使用 value 属性指定地址值</h3>
<s:url value="actionTag">
    <s:param name="message">Hello World!</s:param>
</s:url>

<h3>使用 id 属性,不显示生成的 URL 字符串</h3>
<s:url value="actionTag" id="myUrl">
     <s:param name="message">Hello World!</s:param>
</s:url>
<h3>利用标签 a 引用 url 标签生成的 URL</h3>
<s:a href="%{myUrl}">动态生成的超级链接</s:a>
```

图 6-6 给出了对应的运行结果,读者可以自己尝试着分析一下。

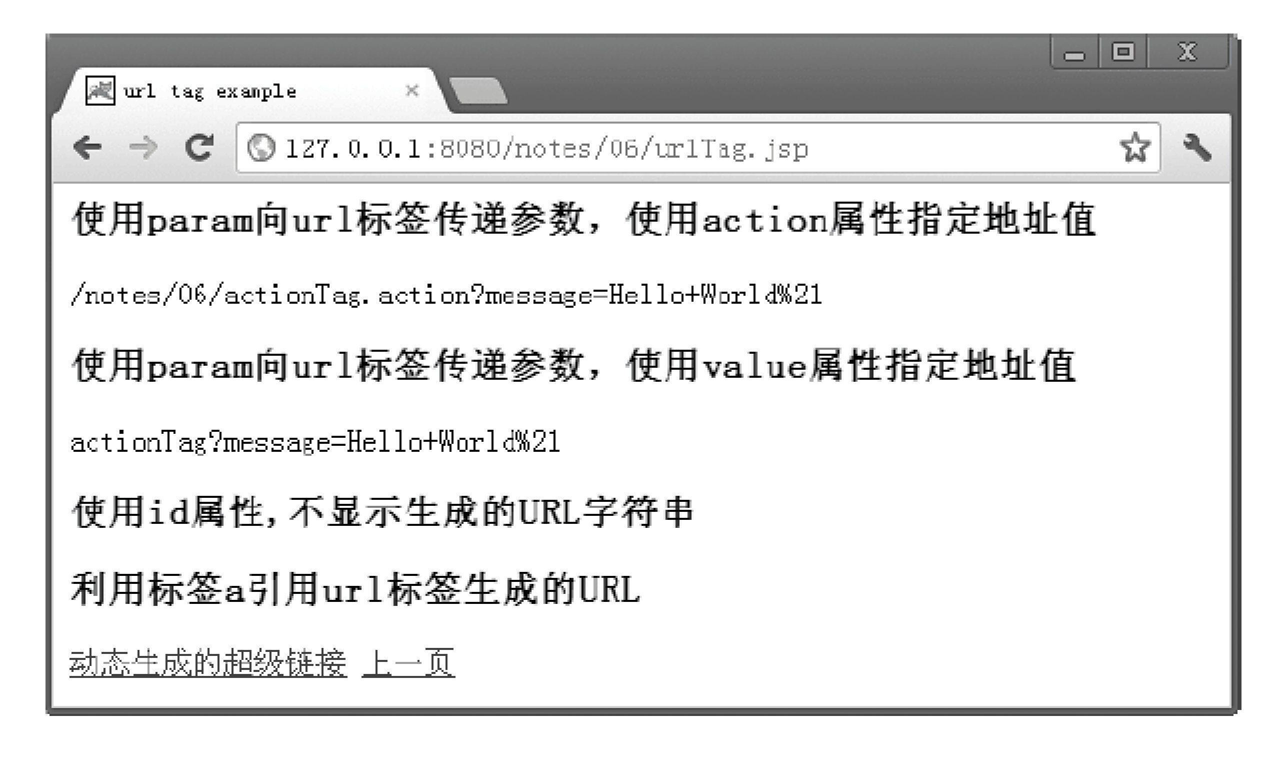


图 6-6　url 标签和 a 标签的运行结果

6.3.9　include 标签

include 标签用于把其他页面或 Servlet 包含到当前的页面中。include 标签和 JSP 的标准动作<jsp:include>非常类似,都是在运行期间将被包含的页面引入到本页。在包含页面时,可以在 include 标签内利用 param 标签向被包含的页面传递请求参数。

需要注意的是,包含页面和被包含页面是相互独立的,不能共享变量。表 6-8 给出了 include 标签的属性。

表 6-8　include 标签的属性

属性名	是否必需	类 型	说　　明
value	是	String	被包含的 JSP 或 Servlet

代码 6-6 给了 include 标签的用法。

代码 6-6　include 标签的用法

```
<!-- includeTag.jsp -->
    <h3>使用 include 标签包含 page1.jsp </h3>
    <s:include value="page1.jsp"></s:include>

    <h3>使用 include 标签包含 page2.jsp </h3>
    <s:include value="page2.jsp">
        <s:param name="message">欢迎学习 Struts2!</s:param>
    </s:include>

<!-- page1.jsp -->
    <%@ page language="java" pageEncoding="UTF-8"%>
    这是 Page1!

<!-- page2.jsp -->
    <%@ page language="java" pageEncoding="UTF-8"%>
    这是 Page2!<br/>
```

```
<!--利用 el 表达式输出传递过来的参数 -->
${param.message}
```

注意：在页面 page2.jsp 中不能使用 property 标签输出传递过来的请求参数 message。如果需要输出的话，则可以使用 JSP 的 el 表达式输出。

图 6-7 给出了代码 6-6 运行的效果。



图 6-7 include 标签的运行效果

6.3.10 date 标签

date 标签用于格式化输出日期型数据，也可以输出当前日期和指定日期之间的时间差。

表 6-9 给出了 date 标签常用的属性。

表 6-9 date 标签常用的属性

属性名	是否必需	类 型	说 明
name	是	String	需要格式化的日期对象
format	否	String	指定日期的格式化样式
nice	否	Boolean	是否显示当前时间与指定时间的差。如果设置为 true，则不再显示指定时间，只显示当前时间与指定时间的差。默认值为 false
var	否	String	用来引用被保存在 Stack Context 的日期对象

当属性 nice 的值设置为 false 时，format 属性将不会起作用。如果 nice 属性和 format 属性都没有设置，则 date 标签将从国际化资源包中查找 struts.date.format 键。找到后，利用该键的值作为日期的格式化样式；如果找不到，就采用 DateFormat.MEDIUM 格式化样式。有关国际化的问题，将在后面的章节讲述。

代码 6-7 演示了 date 标签的用法。

代码 6-7 date 标签的用法

```
<%
    Calendar c=Calendar.getInstance();
    c.set(Calendar.MONTH,c.get(Calendar.MONTH)+1);
    pageContext.setAttribute("calendar",c);
%>
```

```
<h3>格式化输出 date</h3>
    <s:date name="#attr.calendar" format="yyyy年 MM月 dd日" /><br/>

<h3>无格式化输出 date</h3>
    <s:date name="#attr.calendar" /><br/>

    <h3>用 nice 输出当前日期和 calendar 的差</h3>
    <s:date name="#attr.calendar" nice="true" />
```

图 6-8 为上述代码的运行效果。

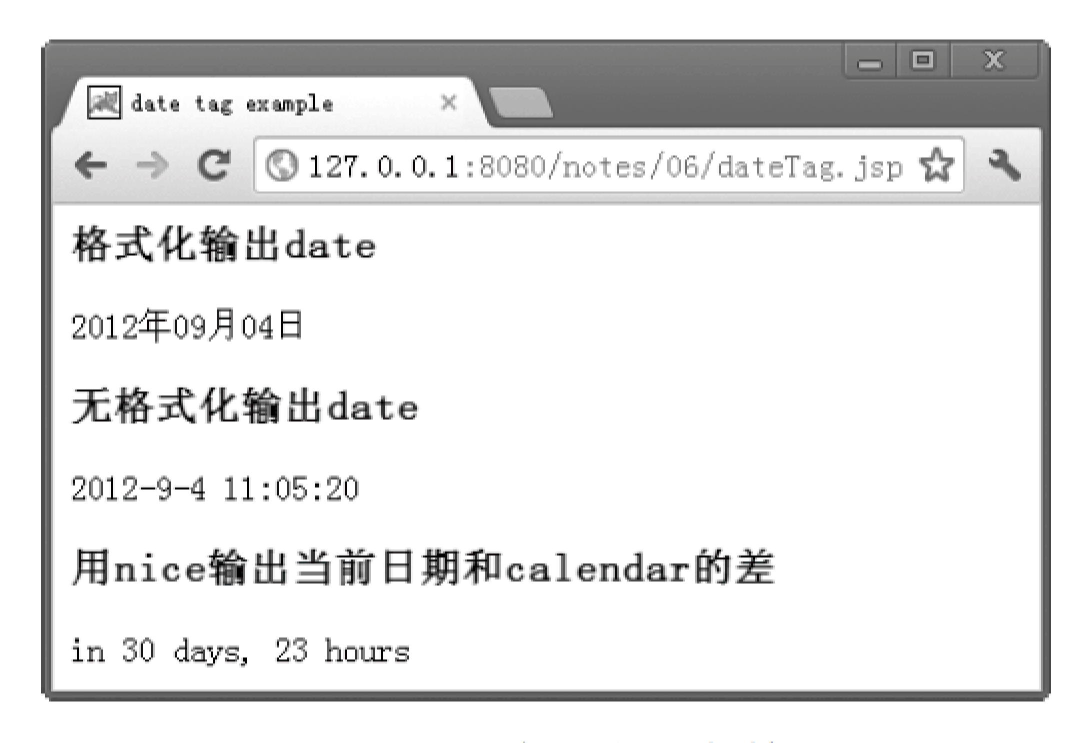


图 6-8　date 标签的运行效果

6.4　控制标签

Struts 2 的控制标签主要关注程序的运行流程。例如，利用 if/elseif/else 标签进行分支控制，利用 iterator 标签实现对集合数据的循环控制等。

6.4.1　if、elseif 和 else 标签

if、elseif 和 else 标签用于对给定的条件进行测试，类似于 Java 语言中的 if、elseif 和 else 关键字。if 标签可以单独使用，也可以和一个或多个 elseif 标签，或者和一个 else 标签一起使用。elseif 标签和 else 标签不能脱离 if 标签单独使用。if 和 elseif 标签都具有一个属性 test，如表 6-10 所示。

表 6-10　if 和 elseif 标签的属性

属性名	是否必需	类 型	说　明
test	是	Boolean	测试条件

代码 6-8 演示了 if、elseif 和 else 标签的用法。

代码 6-8　if、elseif 和 else 标签的用法

```
<h3>测试请求参数输入的数据 </h3>
<s:set name="number" value="#parameters.number[0]"></s:set>
```

```
<s:if test="#number>0">
    您输入的是一个正数!
</s:if>
<s:elseif test="#number==0">
    您输入的数等于 0!
</s:elseif>
<s:else>
    您输入的是一个负数!
</s:else>

<h3>测试用户是否登录 </h3>
    <s:if test="#session.user==null">
        您尚未<s:a href="login.action">登录</s:a>!
    </s:if>
    <s:else>
        欢迎您!
    </s:else>
```

在浏览器中输入"http://127.0.0.1:8080/notes/06/ifTag.jsp?number=-5",可以看到如图 6-9 所示的效果。



图 6-9 if、elseif 和 else 标签的运行效果

6.4.2 iterator 标签

iterator 标签是控制标签中最重要的。可以通过 iterator 标签遍历一个数组、Set、List、Map 和 Iterator 等集合对象。在迭代时,iterator 标签会把要迭代的集合对象中的每一个元素依次压入 Value Stack 和弹出 Value Stack。表 6-11 给出了 iterator 标签常用的属性。

表 6-11 iterator 标签常用的属性

属性名	是否必需	类 型	说 明
begin	否	Integer	指定迭代的起始下标。默认值为 0
end	否	Integer	指定迭代的终止下标
status	否	Boolean	当指定该值时将创建一个 IteratorStatus 实例。默认值为 false
step	否	Integer	指定迭代步长。默认值为 1
value	否	String	指定需要迭代的集合对象
var	否	String	用于引用集合对象中的当前元素

当指定了 status 属性时，将创建一个 IteratorStatus 的对象，该对象保存到 Stack Context 中。利用 IteratorStatus 可以实现获得更多与迭代相关的信息，表 6-12 给出了该对象的属性。

表 6-12 IteratorStatus 对象的属性

属性名	类 型	说 明
index	Integer	正在迭代的元素的下标
count	Integer	已经迭代的元素的个数
first	Boolean	当前被迭代的元素是不是第一个元素
last	Boolean	当前被迭代的元素是不是最后一个元素
even	Boolean	当前被迭代的元素是不是偶数
odd	Boolean	当前被迭代的元素是不是奇数

下面通过例子详细讲述 iterator 标签的用法。首先，创建一个 IteratorAction 类，在该类中封装了一个关于 User 对象的 List，如代码 6-9 所示。

代码 6-9 IteratorAction.java

```
<!-- User.java -->
public class User {
    private int userId;                          //用户 ID
    private String username;                     //用户名
    private String password;                     //密码
    //省略了 username、password 属性的 get 和 set 方法
}

<!-- IteratorAction.java -->
public class IteratorAction extends ActionSupport {
    private List<User> users;
    public List<User> getUsers() {
        return users;
    }

    public String execute(){
        users=new ArrayList<User>();
        for(int i=0;i<5;i++){
            User user=new User();
            user.setUserName("user"+(i+1));
            user.setPassword("password"+(i+1));

            users.add(user);
        }
        return SUCCESS;
    }

}
```

接下来编写 IteratorAction 的结果页面 iteratorTag.jsp，如代码 6-10 所示。

代码 6-10 结果页面 iteratorTag.jsp

```
<h3>迭代输出 List 对象 ,使用 Status 和 var 属性</h3>
    <table border="1" width="300">
        <tr>
            <td>序号</td>
            <td>用户名</td>
            <td>密码</td>
        </tr>
            <s:iterator value="users" status="st" var="user">
                <s:if test="#st.odd">
                    <tr style="background:#efefef">
                </s:if>
                <s:else>
                <tr>
                </s:else>
                <td><s:property value="#st.index+1"/></td>
                <td><s:property value="#user.userName"/></td>
                <td><s:property value="#user.password"/></td>
                </tr>
            </s:iterator>
    </table>

    <h3>迭代输出 List,使用 begin 和 end 属性</h3>
    <s:iterator value="users" begin="1" end="4" var="user2" >
        <s:property value="#user2.userName"/>
    </s:iterator>

    <h3>迭代输出 Map</h3>
    <s:iterator value="#{'one':'red','two':'blue','three':'white'}">
        <s:property/>
    </s:iterator>

    <h3>迭代 Map,分别输出 key 和 value</h3>
    <s:iterator value="#{'one':'red','two':'blue','three':'white'}">
        <s:property value="key"/>:<s:property value="value"/>
    </s:iterator>
```

注意：在使用 var 属性后，被迭代的集合对象会保存到 Stack Context 中，因此在利用 var 引用当前迭代元素时，一定要使用“#”符号。

图 6-10 给出了上述例子的演示效果。

iterator 标签还可以模拟 Java 语言中的 for 循环。例如，下面的语句循环输出 10 行字符串“Hello”。

```
<s:iterator begin="1" end="10">
    Hello<br/>
</s:iterator>
```



图 6-10　iterator 标签的运行效果

6.4.3　append 标签和 merge 标签

append 标签和 merge 标签都是用于将多个集合对象组合成一个新的集合对象，二者的区别在于合并后的集合中元素的顺序不同。

假设有两个集合对象 A 和 B，每个对象包括 3 个元素。使用 append 标签组成的新对象中的元素如下(注意顺序)：

(1) A 的第一个元素

(2) B 的第一个元素

(3) A 的第二个元素

(4) B 的第二个元素

(5) A 的第三个元素

(6) B 的第三个元素

使用 merge 标签组成的新对象中的元素如下(注意顺序)：

(1) A 的第一个元素

(2) A 的第二个元素

(3) A 的第三个元素

(4) B 的第一个元素

(5) B 的第二个元素

(6) B 的第三个元素

append 标签和 merge 标签具有相同的属性，如表 6-13 所示。

表 6-13 append 标签和 merge 标签的属性

属性名	是否必需	类 型	说 明
var	否	String	用于引用组合后的集合对象。指定了该属性后，合并后的集合将放入 Stack Context

代码 6-11 演示了 append 标签和 merge 标签的用法。

代码 6-11 append 标签和 merge 标签的用法

```
<s:set var="list1" value="{'one','two','three'}"></s:set>
<s:set var="list2" value="{'red','blue'}"></s:set>

<h3>用 append将多个列表组合在一起迭代</h3>
<s:append var="allList1">
    <s:param value="#list1"></s:param>
    <s:param value="#list2"></s:param>
</s:append>

<s:iterator value="#alllList1" >
    <s:property/>
</s:iterator>

<h3>用 merge将多个列表组合在一起迭代</h3>
<s:merge var="allList2">
    <s:param value="#list1"></s:param>
    <s:param value="#list2"></s:param>
</s:merge>

<s:iterator value="%{#alllList2}" >
    <s:property/>
</s:iterator>
```

读者可自行分析上述代码在浏览器中的输出结果。

6.4.4 generator 标签

generator 标签用于将一个字符串拆分成若干个字符串构成的集合，可以利用 iterator 标签对拆分结果迭代输出。generator 标签的属性如表 6-14 所示。

表 6-14 generator 标签的属性

属性名	是否必需	类 型	说 明
val	是	String	要拆分的字符串
separator	是	String	拆分时的分隔符
count	否	Integer	拆分后的集合中只容纳前 count 个字符串
converter	否	Converter	用于设置自定义的 Converter 类，该类必须实现接口 org.apache.struts2.util.IteratorGenerator.Converter
var	否	String	用于引用组合后的集合对象。指定了该属性后，合并后的集合将放入 Page Context

代码 6-12 演示了 generator 标签的用法。

代码 6-12 generator 标签的用法

```
<h3>利用 generator 拆分字符串,在内部用 iterator 迭代输出</h3>
<s:generator val="%{'red,blue,green,black'}" separator=",">
    <s:iterator>
        <s:property /><br/>
    </s:iterator>
</s:generator>

<h3>利用 generator 拆分成 3 个字符串,在外部用 iterator 迭代输出</h3>
<s:generator val="%{'red,blue,green,black'}" separator="," count="3"
        var="gen"/>

<s:iterator value="#gen">
    <s:property /><br/>
</s:iterator>

<h3>自定义 Converter 拆分字符串</h3>
<s:generator val="%{'red,blue,green,black'}" separator=","
        converter="%{myConverter}">
    <s:iterator>
        <s:property /><br/>
    </s:iterator>
</s:generator>
```

代码中的最后一个例子用到了一个自定义的 Converter 转换器,将其定义在 GeneratorTagAction.java 中。该 Converter 用方括号将拆分得到的字符串括起来,如代码 6-13 所示。

代码 6-13 GeneratorTagAction.java 中定义 Converter

```
public class GeneratorTagAction extends ActionSupport {
    private Converter myConverter;
    public Converter getMyConverter() {
        return new Converter() {
            public Object convert(String value) throws Exception {
                return "[" + value + "]";
            }
        };
    }

    public String execute() throws Exception {
        return SUCCESS;
    }
}
```

在浏览器中查看上述例子,运行效果如图 6-11 所示。



图 6-11 generator 标签的运行效果

6.4.5 subset 标签

subset 标签用于获取指定集合的子集合。subset 标签的属性如表 6-15 所示。

表 6-15 subset 标签的属性

属性名	是否必需	类 型	说 明
source	否	Collection、Map、Enumeration、Iterator 或 Array	指定源集合。缺省时，取 Value Stack 的栈顶元素作为源集合
start	否	Integer	指定从源集合的第几个元素开始截取，0 代表第一个元素
count	否	Integer	截取的元素的总数
decider	否	Decider	用于设置自定义的 Decider 类（实现 org. apache. struts2. util. SubsetIteratorFilter. Decider 接口）
var	否	String	用于引用截取后的集合对象。指定该属性后，截取后的集合将放入 Page Context

代码 6-14 演示了 subset 标签的用法。

代码 6-14 subset 标签的用法

```
<s:bean var="myDecider" name="example.Decider.OddDecider"></s:bean>
<s:set var="myList" value="{'1','2','3','4','5','6'}"></s:set>

<h3>利用 subset 从列表中截取子集,起始索引为 2,截取 3个元素</h3>
<s:subset source="#myList" start="2" count="3">
    <s:iterator>
        <s:property />
```

```
    </s:iterator>
</s:subset>

<h3>利用 var 属性将截取的子集保存到 pageContext 中</h3>
<s:subset source="myList" count="3" var="subset"></s:subset>
    <%
        Iterator i= (Iterator) pageContext.getAttribute("subset");
        while(i.hasNext()) {
            out.print(i.next()+" ");
        }
    %>

<h3>自定义 Decider,获取列表中的奇数</h3>
<s:subset source="myList" decider="myDecider">
    <s:iterator>
        <s:property />
    </s:iterator>
</s:subset>
```

代码中的最后一个例子使用一个自定义的 Decider 从列表中取出所有的奇数。代码 6-15 给出了自定义的 Decider。

代码 6-15　OddDecider. java

```
import org.apache.struts2.util.SubsetIteratorFilter.Decider;

public class OddDecider implements Decider {
    public boolean decide(Object element) throws Exception {
        String value=element.toString();
        if(Integer.parseInt(value)%2!=0)
            return true;
        return false;
    }
}
```

在浏览器中查看上述例子,运行效果如图 6-12 所示。



图 6-12　subset 标签的运行效果

6.4.6 sort 标签

sort 标签利用设置的比较器来对指定的集合进行排序。sort 标签的属性如表 6-16 所示。

表 6-16 sort 标签的属性

属性名	是否必需	类 型	说 明
source	否	Collection、Map、Enumeration、Iterator 或 Array	指定要排序的集合。缺省时，取 Value Stack 的栈顶元素作为源集合
comparator	是	java. util. Comparator	指定排序时使用的比较器
var	否	String	用于引用排序后的集合对象。指定该属性后，排序后的集合将放入 Page Context

要想使用 sort 标签对集合元素进行排序，首先应该自定义一个比较各个元素的比较器。代码 6-16 给出了一个比较器的例子。

代码 6-16 编写比较器

```
import java.util.Comparator;
public class MyComparator implements Comparator<String> {
    public int compare(String o1, String o2) {
        int diff=o1.length()-o2.length();
        if(diff==0)
            diff=o1.toLowerCase().compareTo(o2.toLowerCase());
        return diff;
    }
}
```

比较器 MyComparator. java 实现了 java. util. Comparator 接口。该比较器用于比较两个字符串的长度。长度相等的字符串，按照字母表比较。代码 6-17 演示了 sort 标签如何利用 MyComparator 对列表进行排序。

代码 6-17 sort 标签的用法

```
<h3>利用 subset 从列表中截取子集,起始索引为 2,截取 3个元素</h3>
<s:sort source="#myList" comparator="myComparator">
    <s:iterator>
        <s:property />
    </s:iterator>
</s:sort>

<h3>利用 var 属性将截取的子集保存到 pageContext 中</h3>
<s:sort source="#myList" comparator="myComparator" var="sortedList"/>

    <%
        Iterator i= (Iterator) pageContext.getAttribute("sortedList");
        while(i.hasNext()) {
            out.print(i.next()+" ");
        }
    %>
```

在浏览器中查看上述例子，运行效果如图 6-13 所示。



图 6-13　sort 标签的运行效果

6.5　表单标签

表单标签用于生成 HTML 元素的标签。表单标签由 form 标签和包装 HTML 表单元素的其他标签。

6.5.1　表单标签的公共属性

Struts 2 中的表单标签对应的类都继承自 UIBean 类，UIBean 基类提供了一组公共的属性。表 6-17 给出了这些标签的通用属性，其中加"*"的属性仅在没有使用 simple 主题时方可使用。与主题有关的内容将在 6.6 节讲述。

表 6-17　表单标签的通用属性

名　　字	类　型	说　　明
cssClass	String	设置 HTML 表单元素的 class 属性
cssStyle	String	设置 HTML 表单元素的 style 属性
title	String	设置 HTML 表单元素的 title 属性
id	String	设置 HTML 表单元素的 id 属性
disabled	String	设置 HTML 表单元素的 disabled 属性
label*	String	设置与 HTML 表单元素关联的 label
labelpostion*	String	设置 HTML 表单元素的 label 的位置(left|top)。默认为 left
required*	Boolean	设置 HTML 表单元素是否为必填项。为 true 时，在 label 上加"*"。默认值 false
requiredpostion*	String	设置必填标记相对于 label 的位置(left|top)。默认值为 right
tabindex	String	设置 HTML 表单元素的 tabindex 属性
name	String	设置 HTML 表单元素的 name 属性。该值对应 Action 类的属性
value	String	设置 HTML 表单元素的 value 属性
key	String	设置 HTML 表单元素的 name、label 和 value，其值来自国际化文件

通用属性中的 name 属性最为重要，该属性和处理表单的 Action 类中的属性相对应。在没有为标签设置 value 属性时，表单元素自动从 Action Stack 中查找名为 name 的

对象(通常为 Action 的属性),利用该对象的值作为该标签的 value 值。

表单还包含了与触发 JavaScript 事件相关的属性,例如 onclick 和 ondblclick 等,这些属性与 HTML 标签的标准 JavaScript 属性完全一致,在此不再赘述。

除了通用属性外,还有些属性与模板和浮动提示有关,如表 6-18 所示。

表 6-18 表单标签的其他属性

名　字	类　型	说　明
templateDir	String	设置模板目录
theme	String	设置主题(simple|xhtml|css_xhtml|ajax)。默认值为 xhtml
template	String	设置模板名称
tooltip	String	设置表单元素的工具提示
tooltipConfig	String	设置与工具提示相关的属性

6.5.2 form 标签

form 标签用于生成 HTML 中的 form 表单。除公共属性外,form 标签的常用属性如表 6-19 所示。

表 6-19 form 标签的常用属性

名　字	类　型	说　明
action	String	设置提交的 Action 的名字
namespace	String	设置 Action 的命名空间
method	String	设置 HTML 表单的 form 属性(get|post)。默认值为 post
enctype	String	在上传文件时,需要将该属性设置为 multipart/form-data
validate	String	在 xhtml/ajax 主题下是否执行客户端验证。默认值为 false
id	String	设置 HTML 表单的 id 属性

在 web.xml 将核心控制器的 url-pattern 属性设置为“/ * ”时,可以省略 action 属性的扩展名(.action),否则不能省略。

例如,语句

```
<s:form action="/notes/login.action"> ... </s:form>
```

将调用/notes 命名空间中名为的 login 的 action。

在浏览器中查看上述语句生成的 HTML 源码如下:

```
<form id="from1" name="from1" action="login.action" method="post">
    <table class="wwFormTable">
    </table>
</form>
```

从中可以看到:

(1) 生成的代码除了一个 form 表单之外,还有一个 table 表格。因此,默认情况下

form 表单自动采用表格进行布局。

(2) 在默认表单的 id 和 name 属性的值时，Struts 2 将 id 和 name 的值设置为 action 的名字。实际上，在表单的 action 属性的缺省时，表单的 action 属性将自动设置为当前 JSP 页面或请求 Action 的名字。另外，在表单中的某个标签的 id 值没有指定时，Struts 2 将其设置为“表单 id_表单标签的 name”。

6.5.3 textfield、password 和 hidden 标签

textfield 标签用于输出一个 HTML 单行文本框，对应 HTML 的<input type="text">。

password 标签用于输出一个 HTML 口令输入控件，对应 HTML 的<input type="password">。

hidden 标签用于输出一个 HTML 隐藏表单元素，对应 HTML 的<input type="hidden">。除公共属性外，hidden 标签没有其他属性。

除了公共属性外，textfield 和 password 标签还具有表 6-20 所示的公共属性。

表 6-20 textfield 和 password 标签的公共属性

名　字	类　型	说　　明
maxlength	String	输入框所能容纳的最大文本长度
readonly	Boolean	设置元素是否是只读。默认值为 false
size	String	设置显示的尺寸

除了上述属性外，password 标签还有一个 showPassword 属性。当该属性的值被设置成 true 时，password 标签将显示密码。

下面的语句演示了 textfield 和 password 标签的用法。

```
<s:form action="/notes/login.action" id="form1">
    <s:textfield name="username" size="40" label="用户名"></s:textfield>
    <s:password name="password" size="40" label="密码"></s:password>
    <s:hidden name="id" value="10"></s:hidden>
</s:form>
```

6.5.4 textarea 标签

textarea 标签用于输出一个 HTML 多行文本框，对应 HTML 的<textarea...>。除了公共属性外，textarea 标签还具有表 6-21 所示的属性。

表 6-21 textarea 标签的属性

名　字	类　型	说　　明
cols	Integer	设置多行文本框的列数
rows	Integer	设置多行文本框的行数
readonly	Boolean	设置多行文本框是否是只读。默认值为 false
size	String	设置多行文本框中的内容是否自动换行

例如：

```
<s:textarea name="content" cols="80" rows="20"></s:textarea>
```

6.5.5 reset 标签

reset 标签用于输出一个重置按钮。除了公共属性外，reset 标签还具有表 6-22 所示的属性。

表 6-22 reset 标签的属性

名字	类　型	说　　明
type	String	用于设置重置按钮的类型(input|button)。默认值为 input

当 type 取值为 input 时，等价于 HTML 的<input type="reset" .../>。此时需要通过 value 属性设置重置按钮的文本。

当 type 取值为 button 时，等价于 HTML 的<button type="reset" .../>。此时需要通过 label 属性设置重置按钮的文本。

例如：

```
<s:reset value="重填"></s:reset>
<s:reset type="button" value="重填"></s:reset>
```

6.5.6 submit 标签

submit 标签用于输出一个提交按钮。除了公共属性外，submit 标签还具有表 6-23 所示的属性。

表 6-23 submit 标签的属性

名　字	类　型	说　　明
action	String	设置 HTML 的 action 属性
method	String	设置使用 Action 的哪个方法处理请求
type	String	设置提交按钮的类型(input|button|image)。默认值为 input
src	String	当 type="image"时，为提交按钮设置图片地址

当 type 取值为 input 时，等价于 HTML 的<input type="submit" .../>。此时需要通过 value 属性设置提交按钮的文本。

当 type 取值为 button 时，等价于 HTML 的<button type="submit" .../>。此时需要通过 label 属性设置提交按钮的文本。

当 type 取值为 image 时，等价于 HTML 的<button type="image" .../>。

Struts 2 为 submit 标签的 name 属性定义了 4 个前缀。

(1) method 前缀：用于指定调用 Action 的哪个方法。

(2) action 前缀：可以改变表单的提交行为。

(3) redirect 前缀：用于将请求重定向到 URL。

（4）redirectAction 前缀：用于将请求重定向到其他 Action。

代码 6-18 给出了一个例子。在这个例子中，表单通过不同的 submit 按钮处理不同的用户请求。请注意代码中的注释部分。

代码 6-18　一个表单中有多个 submit 按钮

```
<s:form action="login.action" id="form1">
    <s:textfield name="username" size="40" label="用户名"></s:textfield>
    <s:password name="password" size="40" label="密码"></s:password>

    <!--使用 form 的 action 属性处理请求 -->
    <s:submit value="登录"></s:submit>

    <!--使用 register.action 取代 form 标签指定的 action -->
    <s:submit value="注册" name="action:register"/>

    <!--使用 login.action 的 forgetPassword 方法处理请求
        等价于<s:submit value="忘记密码 2" method="forgetPassword"/>
     -->
    <s:submit value="忘记密码" name="method:forgetPassword"/>

    <!--使用 redirect 前缀将请求重定向到 google -->
    <s:submit value="搜索" name="redirect:http://www.google.com.hk"/>
    <s:reset type="input" value="重填"/>
</s:form>
```

6.5.7　checkbox 标签

checkbox 标签用于输出一个 HTML 复选框，对应 HTML 的<input type="checkbox">标签。checkbox 标签只能表达 Boolean 值，因此用于处理请求的 Action 类中的对应属性的类型必须是 Boolean 类型。

除了公共属性外，checkbox 标签还具有表 6-24 所示的属性。

表 6-24　checkbox 标签的属性

名　字	类　型	说　　明
fieldValue	String	指定在复选框选中时，实际提交的值(true|false)。默认值为 true

在 fieldValue 的值为 true 时，例如：

```
<s:checkbox name="remember" label="记住密码"></s:checkbox>
```

如果用户在表单上选中了该按钮，则 Action 类的 remember 属性将被设置为 true，否则 remember 的值将为 false。

在 fieldValue 的值为 false 时，例如：

```
<s:checkbox name="remember" fieldValue="false" label="记住密码" />
```

无论用户是否在表单上选中 remember 按钮，Action 类的 remember 属性都会被设置为 false。因此，除非用户知道自己正在做什么，否则不要将 fieldValue 设置为 false。

6.5.8 checkboxlist 和 radio 标签

checkboxlist 标签用于输出一组 HTML 复选框，等价于一组 <input type="checkbox" ...>。

radio 标签用于输出一组 HTML 单选按钮，等价于一组 <type="radio" ...>。

除了公共属性外，checkboxlist 和 radio 标签还具有表 6-25 所示的属性。

表 6-25 checkboxlist 和 radio 标签的属性

名 字	类 型	说 明
list	String	用于生成多选框或单选按钮的集合(Collection、Map、Enumeration、Iterator 或 Array)，必填
listKey	String	用集合对象的哪个属性生成 checkbox 或 radio 的 value 属性
listValue	String	用集合对象的哪个属性生成 checkbox 或 radio 选项的内容

在下面的例子中，演示了生成多选框和单选按钮的不同方法。代码 6-19 给出的 RadioChecklistAction.java 类是这个例子的 Action 类。在这个类中定义了一个静态的 favorites 列表。这样，不管该 Action 类有多少个实例，这个列表只需填充一次即可。请注意掌握这种方法。

代码 6-19 RadioChecklistAction.java 类

```
/* Favorite.java */
public class Favorite {
    int favId;
    String favName;
/* 省略了 getter 和 setter 方法的代码,请读者自行添加 */
}

/* Favorite.java */
public class RadioChecklistAction extends ActionSupport {
    private static List<Favorite> favorites;
    static{
        favorites=new ArrayList<Favorite>();
        String[] s={"爬山", "游泳", "慢跑", "跳舞", "散步"};
        for (int i=0; i<s.length; i++) {
            Favorite f=new Favorite();
            f.setFavId(i);
            f.setFavName(s[i]);
            favorites.add(f);
        }
    }

    public List<Favorite> getFavorites() {
```

```
        return favorites;
    }

    public String execute() throws Exception {
        return SUCCESS;
    }
}
```

代码 6-20 中定义了两组 checkboxlist 和一组 radio。

代码 6-20　checkboxlist 和 radio 标签的用法

```
<s:form>
    <!--用 Map 填充 radio -->
    <s:radio list="#{1:'男',2:'女'}" label="性别" name="gender"
        value="1"></s:radio>

    <!--用 Action 类的集合属性填充 checkboxlist -->
    <s:checkboxlist name="fav" list="favorites" listKey="favId"
        listValue="favName" label="兴趣爱好"></s:checkboxlist>

    <!--用 List 填充 checkboxlist -->
    <s:radio name="education" list="{'高中','本科','硕士','博士','其他'}"
        label="教育程度"></s:radio>
</s:form>
```

6.5.9　select 标签

select 标签用于输出一个 HTML 列表框，等同于 HTML 代码 <select ...> <option ...> ... </option> </select>。

除了公共属性外，select 标签还具有表 6-26 所示的属性。

表 6-26　select 标签的属性

名　字	类　型	说　　明
list	String	用于生成列表框的集合对象(Collection、Map、Enumeration、Iterator 或 Array)，必填
listKey	String	用集合对象的哪个属性生成列表框的 value 属性
listValue	String	用集合对象的哪个属性生成列表框的内容
headerKey	String	在所有的选项前加额外的一个选项作为其标题的 value 属性
headerValue	String	在所有的选项前加额外的一个选项作为其标题的显示文字
emptyOption	Boolean	在标题后是否添加一个空行。默认值为 false
multiple	Boolean	是否表现为多选列表。默认值为 false
size	Integer	列表框中可显示的选项个数。当 size>1 时，表现为列表框

在代码 6-21 给出的例子中，演示了生成列表框的不同方法。

代码 6-21 select 标签的用法

```
<!--用 List 对象生成下拉框 -->
    <s:select list="{'软件工程','计算机科学与技术','数字媒体','通行工程'}"
        name="department" label="选择专业"></s:select>
    <!--用 Map 对象生成下拉框,默认显示为"数据结构" -->
    <s:select list="#{'1':'Java 语言','2':'数据结构','3':'数据库',
        '4':'网络工程'}" name="course" label="选择课程" value="2"></s:select>

    <!--使用集合里放多个 JavaBean 实例来生成下拉框 -->
    <s:select list="#s.schools" listKey="id" listValue="name" name="school"
        label="选择学院" labelposition="top"></s:select>

    <!--多选框 -->
    <s:select list="{'爬山', '游泳', '慢跑', '跳舞', '散步'}" name="favorite"
        size="7" label="选择你最喜欢的运动" headerValue='选择你最喜欢的运动'
        headerKey='-1' emptyOption="true" multiple="true"></s:select>
```

图 6-14 给出了 select 标签在浏览器中的输出效果。



图 6-14 select 标签在浏览器中的输出效果

上述例子中用到一个 SchoolService JavaBean 类为下拉框填充数据。代码 6-22 给出有关 SchoolService 的实现细节。

代码 6-22 填充 Select 标签的 JavaBean 类

```
/* School.java */
public class School {
    private String id;
    private String name;

    public School(){}
    public School(String id,String name){
        this.id=id;
        this.name=name;
    }
```

```
    /* 省略了 get 和 set 方法 */
}

/* SchoolService.java */
public class SchoolService {
    public School[] getSchools(){
        return new School[]{
                new School("001","计算机学院"),
                new School("002","理学院"),
                new School("003","信控学院"),
                new School("004","石化学院")
        };
    }
}
```

6.5.10 optgroup 标签

optgroup 标签通常作为 select 标签的子标签使用，用来生成选项组。optgroup 标签也具有 list、listKey 和 listValue 属性，它们的含义和用法与 select 标签中同名属性的含义和用法完全一样。

在使用 optgroup 标签时，可以使用 label 属性为每个选项组指定一个组名。但是需要注意，这个组名用户是无法通过鼠标选择的。

代码 6-23 所示是 Struts 2 文档中一个关于 optgroup 标签的例子。

代码 6-23 optgroup 标签的用法

```
<s:form>
    <s:select label="My Selection" name="mySelection"
        list="#{'SUPERMAN':'Superman', 'SPIDERMAN':'spiderman'}">
        <s:optgroup label="Adult" list="#{'SOUTH_PARK':'South Park'}" />
        <s:optgroup label="Japanese" list="#{'POKEMON':'pokemon',
            'DIGIMON':'digimon','SAILORMOON':'Sailormoon'}" />
    </s:select>
</s:form>
```

图 6-15 给出了上述代码在浏览器中的输出效果(图中显示的是单击后展开的效果)。

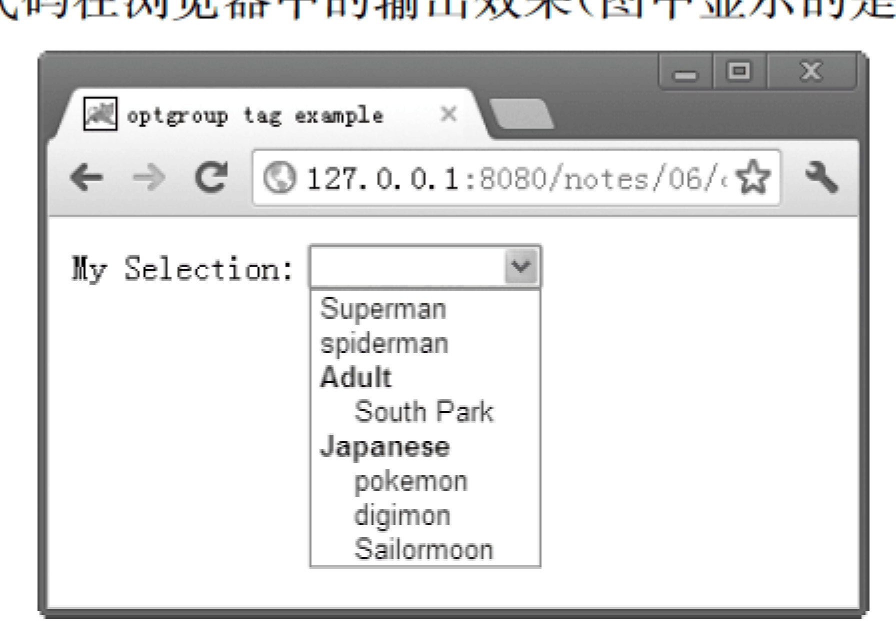


图 6-15 optgroup 标签在浏览器中的输出效果

6.5.11 combobox 标签

combobox 标签用于生成一个文本框和一个下拉框。当下拉框中的选项改变时，它的值将赋给文本框。由于最终提交的是文本框中的值，因此可以使用选项外的其他内容作为提交的数据。

除了公共属性外，combobox 标签也具有 list、listKey、listValue、headerKey、headerValue 和 emptyOption 属性，它们的含义和用法与 select 标签中同名属性的含义和用法完全一样。除此之外，combobox 标签还具有表 6-27 所示的属性。

表 6-27 combobox 标签的属性

名 字	类 型	说 明
maxlength	Integer	文本框中能够输入的文本的最大长度
readonly	Boolean	设置文本框是否为只读，默认值为 false
size	Integer	列表框中可显示的选项个数，当 size>1 时，表现为列表框

下面的代码给出了 combobox 标签的用法。

```
<s:combobox list="{'爬山','游泳','慢跑','跳舞','散步'}"
    name="favoriate"    label="选择你最喜欢的运动" size="20"
    headerKey='-1' headerValue='选择你最喜欢的运动' emptyOption="true">
</s:combobox>
```

6.5.12 updownselect 标签

updownselect 用于输出一个带有上、下移动按钮的列表框。updownselect 标签的实现类是从 select 标签的实现类上扩展而来的，因此它具有 select 标签的所有属性，如 list、listKey 等。除了这些属性和公共属性外，updownselect 标签还具有表 6-28 所示的属性。

表 6-28 updownselect 标签的属性

名 字	类 型	说 明
moveUpLabel	String	设置上移按钮显示的文本
moveDownLable	String	设置下移按钮显示的文本
selectAllLabel	String	设置全选按钮显示的文本
allowMoveUp	Boolean	是否显示向上移动按钮。默认为 true
allowMoveDown	Boolean	是否显示向下移动按钮。默认为 true
allowSelectAll	Boolean	是否显示全部移动按钮。默认为 true

代码 6-24 演示了 updownselect 标签的用法。

代码 6-24 updownselect 标签的用法

```
<s:head/>
<s:form id="frml">
    <!-- 简单的 updownselect -->
```

```
    <s:updownselect list="#{'england':'England','america':'America',
        'germany':'Germany'}" name="prioritisedFavouriteCountries"
        headerKey="-1" headerValue="---请排序---" emptyOption="true" />

    <!--定制的 updownselect -->
    <s:updownselect list="{'爬山', '游泳', '慢跑', '跳舞', '散步'}"
        name="favorite" headerKey="-1"    headerValue="---请排序 ---"
        emptyOption="true" allowMoveUp="true" allowMoveDown="true"
        allowSelectAll="true" moveUpLabel="上移" moveDownLabel="下移"
        selectAllLabel="全选" size="8" />
</s:form>
```

使用 updownselect 标签时,必须配合使用 Struts 2 的 head 标签,因为 updownselect 标签需要引用 Struts 2 的 utils.js,head 标签会自动识别并引入它。

图 6-16 给出了上述代码在浏览器中的输出效果。

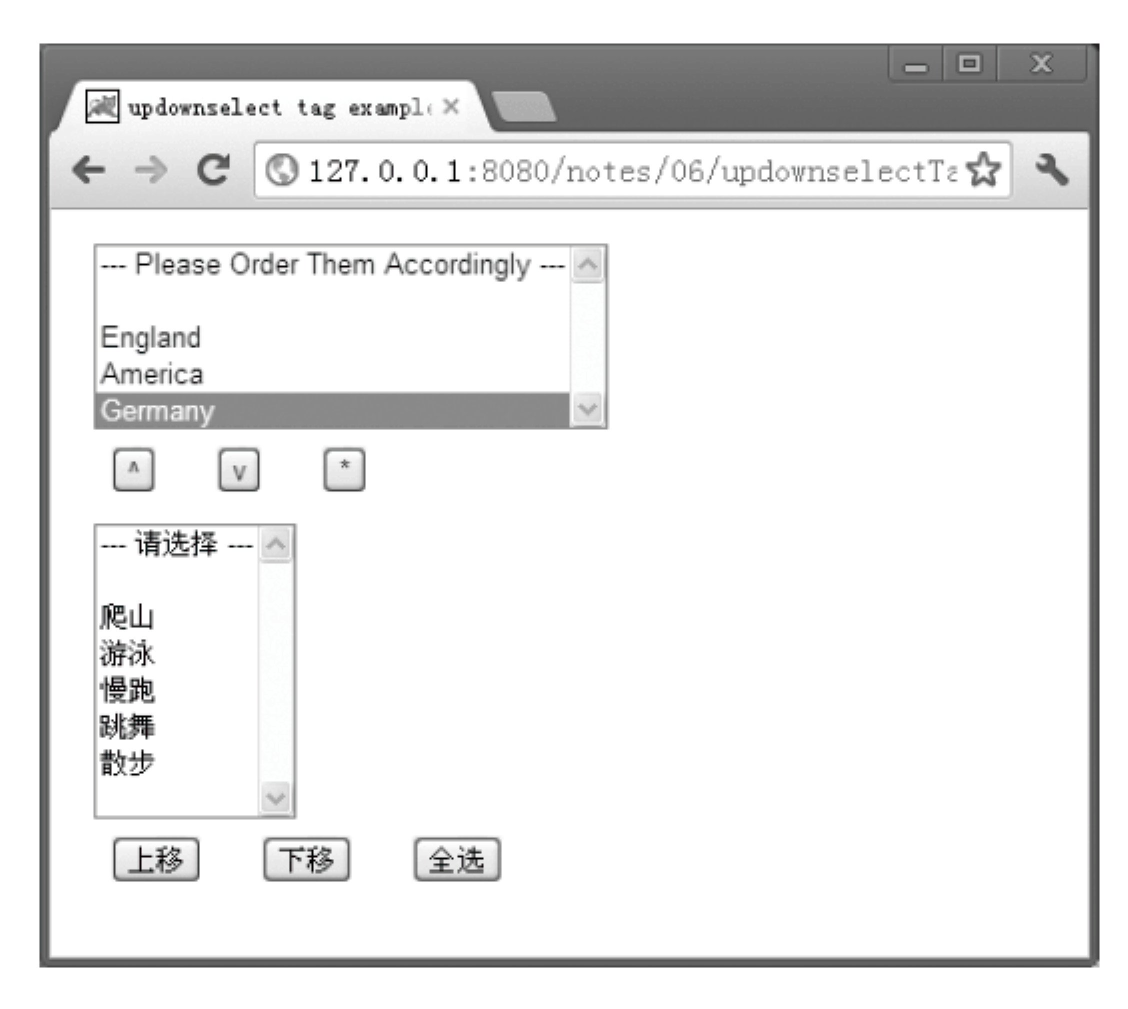


图 6-16 updownselect 标签在浏览器中的输出效果

6.5.13 doubleselect 标签

doubleselect 标签用于输出两个联动的 HTML 列表框。当选择第一个列表框的内容时,第二个列表框的选项随着变化,更新成和第一个列表框选项相关的一组选项。

除了公共属性外,doubleselect 标签还具有表 6-29 所示的属性。

表 6-29 doubleselect 标签的属性

名 字	类 型	说 明
list	String	用于生成第一个列表框的集合对象(Collection、Map、Enumeration、Iterator 或 Array),必填
listKey	String	用集合对象的哪个属性生成第一个列表框的 value 属性
listValue	String	用集合对象的哪个属性生成第一个列表框的内容

续表

名　字	类　型	说　明
headerKey	String	在第一个列表框所有的选项前加额外的一个选项作为其标题的 value 属性
headerValue	String	在第一个列表框所有的选项前加额外的一个选项作为其标题的显示文字
emptyOption	Boolean	在第一个列表框的标题后是否添加一个空行。默认值为 false
size	Integer	第一个列表框中可显示的选项个数
multiple	Boolean	是否表现为多选列表。默认值为 false,对两个列表框均有效
doubleList	String	用于生成第二个列表框的集合对象(Collection、Map、Enumeration、Iterator 或 Array),必填
doubleListKey	String	用集合对象的哪个属性生成第二个列表框的 value 属性
dobuleListValue	String	用集合对象的哪个属性生成第二个列表框的内容
doubleSize	Integer	第二个列表框中可显示的选项个数
doubleName	String	第二个下拉框的 name 属性,必填
doubleValue	String	第二个下拉框的初始选中项

代码 6-25 演示了 doubleselect 标签的第一种用法。

代码 6-25　doubleselect 标签的第一种用法

```
<s:head>
    <s:form>
        <s:doubleselect label="请选择学院"
            name="school"
            doubleName="department"
            doubleValue="'软件工程系'"
            list="{'计算机与通信工程学院','信息与控制工程学院','理学院'}"
            doubleList="(top=='计算机与通信工程学院')?
                {'计算机系','软件工程系'}:
                (top=='信息与控制工程学院'?
                    {'自动化系','测控与仪表系'}:
                    {'应用数学','信息与计算科学'}
                )"
        />
    </s:form>
</s:head>
```

在上面的代码中,doubleselect 标签的 list 属性直接指定了一个 List 集合,里面有三个元素;而 doubleList 属性的值是一个三目运算符,其中的 top 关键字代表第一个下拉框当前被选中的选项值,如果是“计算机与通信工程学院”被选中,则把“计算机系”和“软件工程系”两项添加到下边的下拉框中;如果是“信息与控制工程学院”被选中,把“自动化系”和“测控与仪表系”两项添加到下边的下拉框中;否则,把“应用数学”和“信息与计算科学”两项添加到第二个下拉框中。

代码 6-26 演示了 doubleselect 标签的第二种用法。

代码 6-26　doubleselect 标签的第二种用法

```
<s:head/>
<s:form>
    <s:set name="sd"
    value="#{'计算机与通信工程学院':{'计算机系','软件工程系','通信工程系'},
            '信息与控制工程学院':{'自动化系','测控与仪表系','电子信息工程系'},
            '理学院':{'应用数学','信息与计算科学'}}" />

    <s:doubleselect label="请选择学院" size="3"
            name="school" list="#sd.keySet()"
            doubleList="#sd[top]" doubleSize="3"
            doubleName="department2"/>

</s:form>
```

注意上述代码中 list 属性和 doubleList 属性的赋值。

图 6-17 给出了上述代码在浏览器中的输出效果。



图 6-17　doubleselect 标签在浏览器中的输出效果

6.5.14　optiontransferselect 标签

optiontransferselect 标签用来生成两个左右放置的列表框，通过按钮可以控制选项在两个列表框之间移动。

optiontransferselect 标签的实现类是从 doubleselect 标签的实现类上扩展而来的，因此，它具备表 6-29 中列出的所有与 doubleselect 标签有关的属性(multiple 属性除外)。除了这些属性和公共属性外，optiontransferselect 标签还具有表 6-30 列出的属性。

表 6-30　optiontransferselect 标签的属性

名　字	类　型	说　明
multiple	Boolean	左边的列表框是否允许多选。默认值为 false
doubleMultiple	Boolean	右边的列表框是否允许多选。默认值为 false
leftTitle	String	左边列表框的标题

续表

名　字	类　型	说　明
rightTitle	String	右边列表框的标题
addToLeftLabel	String	向左移动按钮上显示的文本
addTRightLabel	String	向右移动按钮上显示的文本
addToAllLeftLabel	String	全部移动到左边按钮上显示的文本
addToAllRightLabel	String	全部移动到右边按钮上显示的文本
selectAllLabel	String	选择全部按钮上显示的文本
leftUpLabel	String	左边下拉框上移按钮上显示的文本
leftDownLabel	String	左边下拉框下移按钮上显示的文本
rightUpLabel	String	右边下拉框上移按钮上显示的文本
rightDownLabel	String	右边下拉框下移按钮上显示的文本
allowAddToleft	Boolean	是否出现向左移动按钮。默认值为 true
allowAddToRight	Boolean	是否出现向右移动按钮。默认值为 true
allowSelectAll	Boolean	是否出现选择全部按钮。默认值为 true
allowUpDownOnLeft	Boolean	是否出现左边列表框的上下移动按钮。默认值为 true
allowUpDownOnRight	Boolean	否出现右边列表框的上下移动按钮。默认值为 true

代码 6-27 演示了 optiontransferselect 标签的用法。

代码 6-27　optiontransferselect 标签的用法

```
<s:head/>
<s:form>
    <s:optiontransferselect
        label="请选择课程"
        name="allCourse"
        leftTitle="全部课程:"
        rightTitle="已选择课程"
        list="{'网络工程','离散数学','操作系统'}"
        multiple="true"
        addToLeftLabel="左移"
        selectAllLabel="全选"
        addAllToLeftLabel="全部左移"
        headerKey="courseKey"
        headerValue="---全部课程---"
        emptyOption="true"
        doubleList="{'Java程序设计', '数据结构','面向对象程序设计'}"
        doubleName="enBook"
        doubleHeaderKey="enKey"
        doubleHeaderValue="---选择课程---"
```

```
            doubleEmptyOption="true"
            doubleMultiple="true"
    />
    </s:form>
```

图 6-18 给出了上述代码在浏览器中的输出效果。



图 6-18 optiontransferselect 标签在浏览器中的输出效果

6.5.15 其他 UI 标签

除了上面介绍的标签之外,还有一些比较简单的标签,也具有上述通用属性。

(1) file 标签:输出一个 HTML 文件选择框,等价于<input type="file">。将在后面的文件上传部分讲述该标签。

(2) label 标签:输出一个文本值,也可以为其他标签提供 label 值。

(3) head 标签:该标签不产生任何表单项,但是某些标签可能需要引用 JavaScript。head 标签能够自动识别页面需要 Struts 2 的哪些 js 文件,并自动导入。

(4) token 标签:用于输出两个隐藏的表单字段,用于防止表单的重复提交。使用时,需要启用 TokenInterceptor 或者 TokenSessionInterceptor 拦截器。

6.6 actionerror、actionmessage 和 fielderror 标签

actionerror、actionmessage 和 fielderror 标签都是用于输出消息的,不同的是 actionerror 标签输出的是 Action 的错误信息,actionmessage 标签输出的是 Action 的一般性消息,而 fielderror 标签输出的是和 action 属性有关的错误。程序员可以在 Action 类中或者验证器类中添加这些信息,然后在 JSP 页面中显示输出。

先看一个 Action 类,如代码 6-28 所示。

代码 6-28 MessageAction. java

```
public class MessageAction extends ActionSupport {
    public String execute()throws Exception{

        //添加 action 一般性消息
        this.addActionMessage("第一条 Action 消息");
        this.addActionMessage("第二条 Action 消息");

        //添加 action 错误消息
        this.addActionError("第一条 Action 错误消息");
        this.addActionError("第二条 Action 错误消息");

        //添加 field 级错误信息
        this.addFieldError("field1", "第一条 field 级错误");
        this.addFieldError("field2", "第二条 field 级错误");

        return SUCCESS;
    }
}
```

上述代码利用了 ActionSupport 类提供的 addActionMessage()和 addActionError()向 Action 级添加一般性信息和错误信息，利用 addFieldError()方法为特定属性添加错误信息。

代码 6-29 给出的 message. jsp 页面显示了上述信息。

代码 6-29 message. jsp 页面

```
<s:form>
    <h3>输出 Action 信息</h3>
    <s:actionmessage/>

    <h3>输出 Action 错误</h3>
    <s:actionerror/>

    <h3>输出所有字段错误</h3>
    <s:fielderror></s:fielderror>

    <h3>输出 field1 字段错误</h3>
    <s:fielderror>
        <s:param>field1</s:param>
    </s:fielderror>
</s:form>
```

注意：fielderror 标签和 param 标签结合可以显示特定属性的错误。在浏览器中访问 MessageAction，可以看到如图 6-19 所示的输出。



图 6-19　actionerror、actionmessage 和 fielderror 标签在浏览器中的输出效果

6.7　模板和主题

先来看一个登录页面的代码片段。

```
<s:form action="login.action" id="from1">
    <s:textfield name="username" size="40" label="用户名"/>
    <s:password name="password" size="40" label="密码"></s:password>
    <s:checkbox name="remember" label="记住密码"></s:checkbox>
    <s:submit value="登录"></s:submit>
    <s:reset type="input" value="重填"></s:reset>
</s:form>
```

在浏览器中观察上述代码的输出效果，如图 6-20 所示。



图 6-20　采用默认主题的登录页面

之所以能够看到这样的布局效果，是因为 Struts 2 在将上述表单转换成 HTML 代码时，利用一种叫做主题的技术进行渲染。

主题和模板是 Struts 2 UI 标签的核心，模板是一个 UI 标签的外在表示形式。例如，上面的例子中使用了 checkbox 标签，Struts 2 根据对应的 checkbox 模板生成一个有模板特色的复选按钮。Struts 2 支持 FreeMarker、Velocity 和 JSP 模板引擎，默认情况下使用的是 FreeMarker 模板。也就是说，Struts 2 提供的 UI 标签的模板都是使用 FreeMarker 编写的。所有 UI 标签对应的模板形成了一个主题。

当要改变 UI 标签的外观时，可以直接设置该 UI 标签需要使用的模板，也可以设置

该 UI 标签使用的主题。通常，不推荐直接设置模板属性，而是选择特定主题。可以通过以下几种方法设置一个 UI 标签的主题。

(1) 通过设定某个 UI 标签上的 theme 属性来指定主题。

(2) 通过设定某个 UI 标签外围的 Form 标签的 theme 属性来指定主题。

(3) 通过取得 page 会话范围内以 theme 为名称的属性来确定主题。

(4) 通过取得 request 会话范围内以 theme 为名称的属性来确定主题。

(5) 通过取得 session 会话范围内以 theme 为名称的属性来确定主题。

(6) 通过取得 application 会话范围内以 theme 为名称的属性来确定主题。

(7) 通过取得名为 struts.ui.theme 的常量(默认值是 xhtml)来确定主题。该常量可以在 struts.properties 文件或者 struts.xml 文件中确定。

对于上面介绍的几种指定 UI 标签主题的方式，其优先级与上述顺序一致。

Struts 2 提供了四种主题。

(1) simple 主题：simple 主题将 UI 标签直接转换成对应的 HTML 元素，不加任何修饰。在使用 simple 主题时，UI 标签的部分属性将不起作用。

(2) xhtml 主题：xhtml 主题是 Struts 2 的默认主题，它采用一个具有两列的 table 布局 form 表单。每个 UI 标签占据表中的一行。其中 UI 的 label 标签显示的属性占据第一列，UI 标签对应的 HTML 元素占据第二列。在测试上述表单的代码时，可以通过浏览器提供的查看源码功能查看输出到客户端的源码。

为使图 6-20 中的表单布局更美观一些，可以将代码修改成下述形式。修改后，“登录”和“重填”按钮位于同一行，如图 6-21 所示。



图 6-21 修改标签主题后的登录页面

```
<s:form action="login.action" id="from1">
    <s:textfield name="username" size="40" label="用户名"/>
    <s:password name="password" size="40" label="密码"></s:password>
    <s:checkbox name="remember" label="记住密码"></s:checkbox>
    <tr>
        <td></td>
        <td>
            <s:submit value="登录" theme="simple"></s:submit>
            <s:reset type="input" value="重填" theme="simple"></s:reset>
        </td>
    </tr>
</s:form>
```

(3) css_xhtml 主题：该主题类似于 xhtml 主题，但是采用 CSS 和 DIV 实现布局和排版。

(4) ajax 主题：在 xhtml 主题的基础上增加了一些高级的 Ajax 功能。该主题自提出到现在，一直处于实验阶段。

在实际开发时，有经验的开发人员一般都会使用 simple 主题，然后通过定义自己的 CSS，利用 DIV 实现页面的布局。

6.8　案例 3：用 Struts 2 标签库改写留言板的视图

本节将利用 Struts 2 标签改写留言板程序的各个 JSP 页面。

1. 公共文件 header.jsp

公共文件 header.jsp 用于输出程序的 LOGO 以及欢迎信息，如代码 6-30 所示。session 中的 user 对象是 login.action（见第 5 章代码 5-23）在完成对用户的身份验证后保存起来的。

代码 6-30　公共文件 header.jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<div class="header">
    <div class="welcomePanel">
        <s:if test="null!=#session.user.userName"><!--测试用户是否登录 -->
                欢迎 <s:property value="#session.user.userName" />回来
            <img alt="" src="images/split.jpg">
            <a href="logout.action">登出</a>
        </s:if>
    </div>
</div>
```

注意：由于 header.jsp 需要被包含在 index.jsp、detail.jsp 和 post.jsp 中，而这些文件中已经引入了 Struts 2 标签库，因此可以省略 taglib 指令。当省略时，其他页面只能使用语句。

```
<%@ include file="header.jsp" %>
```

来包含 header.jsp，而不能使用<jsp:include>指令或者 Struts 2 的 include 标签，否则登录用户名将无法显示出来。

2. 留言列表 index.jsp

留言列表 index.jsp 是 listNotes.action 在返回结果码为 SUCCESS 时的视图页面，它通过 iterator 标签迭代输出 listNotes 实例的 notes 列表，如代码 6-31 所示。

代码 6-31　留言列表 index.jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<html>
    <head>
        <title>留言列表</title>
    <link rel="stylesheet" type="text/css" href="style/main.css">
```

```
    </head>
    <body>
        <%@ include file="header.jsp" %>
        <a href="addNote.action"><img src="images/book_write.png"/></a>
        <div class="ui-widget-header" style="text-align:center;">留言列表
        </div>
        <div class="col1">留言人</div>
        <div class="col2">主题 </div>
            <!--迭代输出 Action 的 notes 属性 -->
            <s:iterator value="notes" var="note">
                <div class="clear"></div>
                <div class="col1"><!-用属性标签输出留言人的姓名 -->
                    <s:property value="#note.user.userName"/>
                </div>
                <div class="col2">
                <!-用 a 标签构造留言内容的 URL,注意其中"%"的用法 -->
                <s:a href="detail.action?noteId=%{#note.noteId}">
                    <s:property value="#note.title"/></s:a>
                 </div>
            </s:iterator>
    </body>
</html>
```

3. 留言页面 post.jsp

post.jsp 页面需要为用户提供一个书写新留言的接口，这里使用 Struts 2 的 UI 标签编写表单。Struts 2 默认的主题为 xhtml，但由于目前很少采用表格形式进行布局，所以这里将表单的主题设置为 simple，并采用 CSS 和 DIV 进行布局。代码 6-32 给出了 post.jsp 的样例。

代码 6-32　留言页面 post.jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<html>
    <head>
        <title>新留言</title>
        <link rel="stylesheet" type="text/css" href="style/main.css">
        <!-- kindeditor 配置 -->
        <script charset="utf-8" src="kindeditor/kindeditor-min.js"></script>
        <script>
            var editor;
            KindEditor.ready(function(K) {
                editor=K.create('textarea[name="note.content"]'
            ); });
        </script>
    </head>
```

```
    <body>
        <%@ include file="header.jsp" %>
        <div class="ui-widget-header" style="text-align:center;">新留言
        </div>
            <s:form action="addNote.action" theme="simple">
                <div class="col1">标题</div>
                <div class="col2">
                    <s:textfield name="note.title" size="50">
                    </s:textfield>
                </div>
                <div class="clear"></div>
                    <div class="col1">内容</div>
                    <div class="col2">
                        <s:textarea rows="20" cols="60" name="note.content">
                        </s:textarea>
                     (少于 1000 字)
                    </div>
                    <div class="col2">
                        <s:submit value="确定"></s:submit>
                        <s:reset value="重填"></s:reset>
                    </div>
            </s:form>
    </body>
</html>
```

上述代码中使用了 kindeditor html 在线编辑器将 textarea 标签替换为一个富文本输入框，使得用户可以通过所见即所得的方式对留言内容格式化，从而提供更为友好的输入界面。在使用 kindeditor 时，需要首先从 http://www. kindsoft. net/获取 kindeditor 压缩包，将其解压并存放在 webroot 的 kindeditor 目录下，然后在需要为用户提供富文本输入的页面，使用如下格式将一个 HTML 的 textarea 标签转换成 HTML 编辑器。

```
<script charset="utf-8" src="kindeditor/kindeditor-min.js"></script>
    <script>
        var editor;
        KindEditor.ready(function(K) {editor=
            K.create('textarea[name="note.content"]'); });
    </script>
```

注意：其中的 name 属性值一定要和 textarea 标签的 name 属性值相同。

图 6-22 给出了留言页面的浏览效果。

读者可以尝试着利用 xhtml 主题对留言页面进行布局。另外，由于 login. jsp 页面和 detail. jsp 页面的布局比较简单，这里不再给出，读者可以尝试着自己完成。



图6-22 留言页面

同步训练

1. 利用Struts 2标签为留言板程序编写注册页面、登录页面login.jsp和显示留言页面detail.jsp,要求分别应用simple主题和xhtml主题。

2. 使用Struts 2标签编写站内短信系统的视图页面。

Chapter 7 第7章 拦 截 器

7.1 Struts 2拦截器

拦截器(Interceptor)是 Struts 2 最重要的特性之一,它类似于 Servlet 的过滤器以及 Java 中的 Proxy 类。通过拦截器,程序员可以在 Action 方法执行前后以及在 Result 执行之后进行一些功能处理,例如身份验证、日志处理等。读者可以回想一下第 5 章给出的 Struts 2 体系结构图中关于拦截器的部分。

与 Servlet 过滤器不同的是,拦截器的功能更为强大。拦截器与 Servlet 的 API 无关,并且能够访问 Value Stack 中的内容。事实上,Struts 2 框架提供的很多特性都是通过拦截器实现的,例如 Action 属性值的注入、异常处理、文件上传、生命期回调和输入校验等。

Struts 2 中预定义了大约 30 多个拦截器,这些拦截器都定义在 struts-default. xml 文件的 struts-default 包内,程序员可以在 struts. xml 中直接引用这些拦截器。下面简单介绍比较常见的几个拦截器。

(1) chain 拦截器:将前一个执行结束的 Action 的属性复制到当前的 Action 中,完成了 result 映射中的 chain 结果类型的映射。

(2) checkbox 拦截器:添加了对 checkbox 自动处理代码。当检测到 checkbox 未选择时,将自动将其设定为 false。

(3) conversionError 拦截器:将存储在 ActionContext 中的类型转化错误信息取出来,添加到相应的 Action 的字段错误中。

(4) execption 拦截器:提供异常处理,允许程序员把一个异常映射到一个结果码。

(5) execAndWait 拦截器:当一个 Action 执行时间很长时,该拦截器可以让 Action 到后台执行,并提供给用户一个友好的进度信息。

(6) fileUpload 拦截器:用于提供对文件上传的支持。

(7) i18n 拦截器:用于支持国际化。它能够将当前会话选择的 locale 放到用户的 session 中。

(8) logger 拦截器:用于在日志信息中输出要执行的 Action 信息。程序员在调试程序时,能够很快地定位到对应的 Action。

(9) modelDriven 拦截器:当 Action 实现 ModelDriven 接口时,它将 getModel()取得的模型对象存入 Value Stack。

(10) params 拦截器：负责将请求参数映射到 Action 的同名属性上。

(11) scope 拦截器：将 Action 状态保存到 session 或 application 范围。

(12) servletConfig 拦截器：提供 Action 直接对 Servlet API 的访问，把 Servlet API 的对象注入 Action，包括：ServletRequestAware、ServletResponseAware、ParameterAware、ApplicationAware 和 SessionAware。

(13) validation 拦截器：调用验证框架读取 xxxAction-validation. xml 文件，并执行在这些文件中声明的校验。

(14) timer 拦截器：输出 ActionInvocation 余下部分执行的时间，便于寻找性能瓶颈。

(15) token 拦截器：用于检查传到 Action 中的 token 的有效性，用于避免表单重复提交。

(16) tokenSession 拦截器：功能与 token 类似，但它把提交的数据保存到 session 中。引用该拦截器可以防止页面重复提交。

7.2 自定义拦截器

除了可以使用 Struts 2 提供的拦截器外，程序员还可以编写自己的拦截器。自定义拦截器时，可以实现 com. opensymphony. xwork2. interceptor. Interceptor 接口。

Interceptor 接口中定义了三个方法。

(1) void init()：该方法在对象初始化时被调用，用于初始化拦截器需要的资源。

(2) void destroy()：用于释放拦截器占用的资源，类似于 C++ 中的析构函数。

(3) String interceptor(ActionInvocation invocation)：interceptor()方法是真正实现拦截器功能的方法，自定义拦截器主要就是实现这个方法。该方法具有唯一的参数 invocation，它是 ActionInvocation 的一个实例，第 5 章中曾经说过，ActionInvocation 负责调度 Interceptor、Action 和 Result，它握有 Action Context，因此能够很方便地访问 Value Stack 中的内容。图 7-1 所示为 Struts 2 官方文档中给出的一个关于拦截器、Action、Result 和 Action Context 之间的关系。

代码 7-1 给出一个自定义拦截器的例子。

代码 7-1 自定义拦截器

```
package example.Interceptor;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.Interceptor;

public class TestInterceptor implements Interceptor {
    private String idName;
    /* 省略了 get 和 set 方法 */

    public String intercept(ActionInvocation arg0) throws Exception {
```

```
            System.out.println(idName+ ":Action还没有执行!");
            String resultCode= arg0.invoke();
            System.out.println(idName+ ":Action已经执行结束!");
            return resultCode;
        }

        public void destroy() {
            System.out.println(idName+ ":拦截器结束");
        }

        public void init() {
            System.out.println(idName+ ":拦截器开始");
        }
    }
```

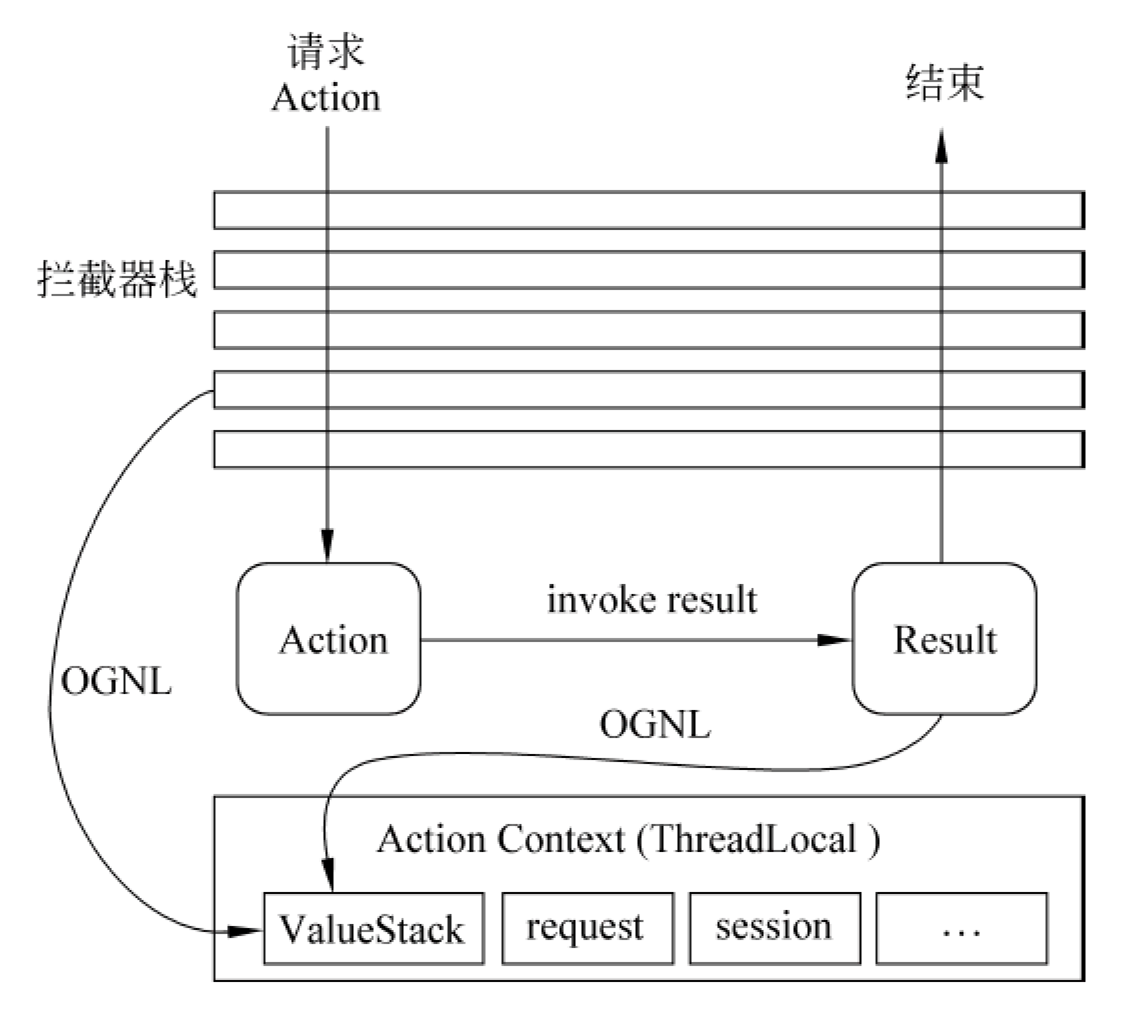


图 7-1　拦截器、Action、Result 和 Action Context 之间的关系

在 interceptor()方法中的语句

```
String resultCode= arg0.invoke();
```

负责将用户请求向下传递，并获得后续操作返回的结果码。当这条语句被删除时，后续的拦截器、Action 和 Result 不会被执行。在有些情况下，拦截器可能希望阻止一个 Action 执行，例如重复提交的情况下或者校验没有通过的情况下，此时可以不让拦截器去执行 arg0. invoke()。

除了实现 Interceptor 接口外，还可以扩展 Interceptor 的默认实现类。

```
com.opensymphony.xwork2.interceptor.AbstractInterceptor
```

如果希望自定义的拦截器只拦截 Action 中指定的方法，或者不拦截某些方法，需要使用 Struts 2 提供的一个特殊拦截器抽象基类。

```
com.opensymphony.xwork2.interceptor.MethodFilterInterceptor
```

很多拦截器都是从 MethodFilterInterceptor 上扩展来的，例如 ValidationInterceptor、TokenInterceptor 和 ParametersInterceptor 等。该拦截器具有两个参数。

（1）excludeMethods：要排除的方法列表，多个方法时用逗号“，”分隔。

（2）includeMethods：要拦截的方法列表，多个方法时用逗号“，”分隔。

7.3 拦截器的配置和使用

Struts 2 拦截器需要在 struts.xml 中声明。在配置 struts.xml 时，只要 package 是从 struts-default 继承，package 就会自动拥有 struts-default.xml 中的所有配置，包括默认的拦截器。

自定义拦截器的声明应该使用 interceptor 元素，并将 interceptor 元素放在 interceptors 元素里。格式如下：

```
<interceptors>
    <interceptor name="拦截器名 1 " class="拦截器的实现类 1 "/>
    <interceptor name="拦截器名 2 " class="拦截器的实现类 2 "/>
    …
</interceptors>
```

如果要为某个 Action 应用声明好的拦截器，需要在 Action 元素中使用 interceptor-ref 元素。

例如，现在有一个 Action 类，如代码 7-2 所示，可以为该 Action 添加一个拦截器 TestInterceptor（见代码 7-1）。在 strus2.xml 中，按照代码 7-3 所示方法进行配置。

代码 7-2 示例 Action 类

```
package example.action;
import com.opensymphony.xwork2.ActionSupport;
public class ICAction extends ActionSupport {
    public String execute(){

        System.out.println("正在执行 Action.");
        return SUCCESS;
    }
}
```

代码 7-3 声明和使用拦截器

```
<package name="default" namespace="/" extends="struts-default">
        <!--声明一个拦截器 ic1 -->
        <interceptors>
            <interceptor name="ic1"
                class="example.Interceptor.TestInterceptor">
            </interceptor>
```

```
        </interceptors>

        <!--在 Action 中应用拦截器,并为拦截器注入参数-->
        <action name="ic" class="example.action.ICAction">
            <result type="redirectAction">list</result>
            <interceptor-ref name="ic1">
                <param name="icName">ic1</param>
            </interceptor-ref>
        </action>
</package>
```

上述代码在声明拦截器的同时,还利用 param 元素将拦截器的 icName 属性赋值为"ic1"。

当一个 Action 需要使用多个拦截器时,可以将这些拦截器组合在一起,组成一个拦截器栈,然后在 Action 中利用 interceptor-ref 元素直接引用拦截器栈。拦截器栈的声明是在 interceptors 元素中使用 interceptor-stack 完成的,格式如下:

```
<interceptors>
    <interceptor-stack name="拦截器栈名">
        <interceptor-ref="拦截器或拦截器栈名 1" />
        <interceptor name="拦截器或拦截器栈名 2"/>
        ...
    <interceptor-stack>
</interceptors>
```

代码 7-4 给出了拦截器栈配置和引用的方法。

代码 7-4 声明和引用拦截器栈

```
<package name="ic" namespace="/" extends="struts-default">
        <!--声明一个拦截器 ic1 -->
        <interceptors>
            <interceptor name="ic1"
                class="example.Interceptor.TestInterceptor">
            </interceptor>

            <!--声明一个拦截器栈 ic -->
            <interceptor-stack name="ic">
                <interceptor-ref name="ic1">
                    <param name="icName">ic1</param>
                </interceptor-ref>
                <interceptor-ref name="ic1">
                    <param name="icName">ic2</param>
                </interceptor-ref>
                <!--引用默认拦截器栈 -->
                <interceptor-ref name="defaultStack"></interceptor-ref>
            </interceptor-stack>
        </interceptors>
```

```
            <!-- 在 Action 中应用拦截器 -->
            <action name="ic" class="example.action.ICAction">
                <result>/success.jsp</result>
                <interceptor-ref name="ic"></interceptor-ref>
            </action>
    </package>
```

上述代码配置了一个拦截器栈 ic，栈中包括两个拦截器和一个拦截器栈 defaultStack。

(1) 在声明拦截器时可以同时为拦截器的参数赋值。如果在声明和应用拦截器时都为同一个参数赋了值，起作用的将是应用拦截器时的参数值。

(2) defaultStack 是 Struts 2 框架的默认拦截器栈，包含 exception、alias 等 18 个拦截器。当为一个 Action 配置了 interceptor-ref 元素时，如果所引用的拦截器栈中没有包含对 defaultStack 的引用，Struts 2 无法为 Action 完成属性映射等功能。因此，一般情况下，在为一个 Action 配置自定义的拦截器或拦截器栈时，一定不要忘记引用 defaultStack。

(3) 位于拦截器栈中的拦截器的顺序非常重要，它直接决定了拦截器执行的顺序。例如，启动 Tomcat 服务器，在浏览器中请求 ic. action，而后关闭 Tomcat，可以从控制台得到如图 7-2 所示的输出(省略了 Tomcat 其他输出信息)。

```
ic1:拦截器开始
ic2:拦截器开始
…
ic1:Action还没有执行!
ic2:Action还没有执行!
正在执行Action.
ic2:Action已经执行结束!
ic1:Action已经执行结束!
…
ic1:拦截器结束
ic2:拦截器结束
```

图 7-2　拦截器运行输出

为了使读者看得更清晰，在图 7-3 给出了 ic. action 执行时拦截器、Action 类和 Result 的序列图。

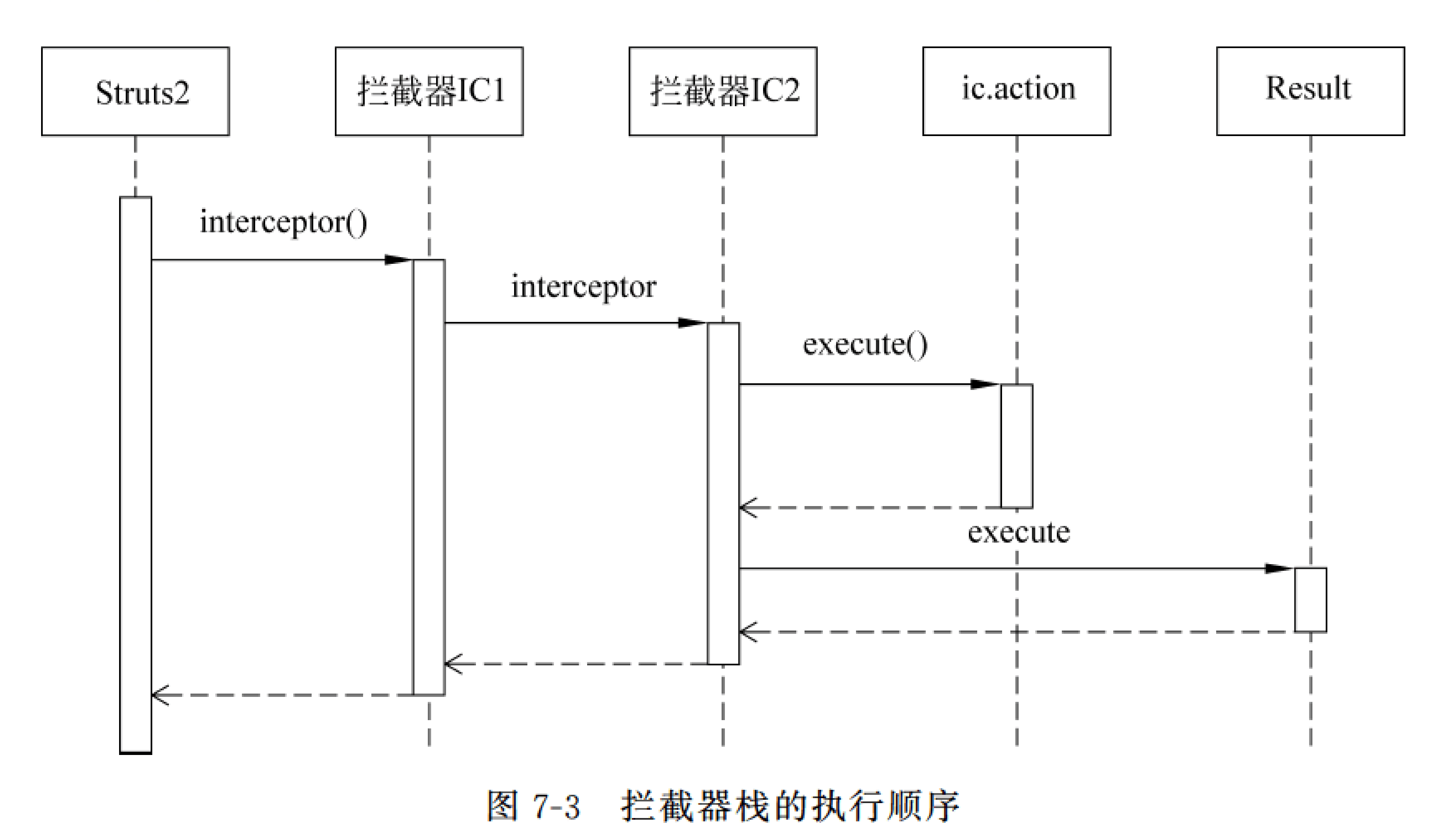


图 7-3　拦截器栈的执行顺序

根据 ic. action 执行情况，可以得知：

(1) Struts 2 框架装在加载 struts.xml 时，执行拦截器的 init()方法；

(2) 同一拦截器栈中，先配置的拦截器先执行，后配置的拦截器后执行；

(3) Struts 2 框架在卸载前、后执行拦截器的 destroy()方法。

如果多个 Action 都要引用相同的拦截器，可以使用 default-interceptor-ref 元素定义一个默认的拦截器或拦截器栈，这样就不需要为每个 Action 都指定拦截器引用了。

代码 7-5 给出了默认拦截器和拦截器栈的声明方法。

代码 7-5 配置默认的拦截器栈

```
<package name="ic2" namespace="/" extends="struts-default">
    <!-- 声明一个拦截器 ic1 -->
    <interceptors>
        <interceptor name="ic1"
            class="example.Interceptor.TestInterceptor">
        </interceptor>

        <!-- 声明一个拦截器栈 ic -->
        <interceptor-stack name="ic">
            <interceptor-ref name="ic1">
                <param name="icName">ic1</param>
            </interceptor-ref>

            <!-- 引用默认拦截器栈 -->
            <interceptor-ref name="defaultStack"></interceptor-ref>
        </interceptor-stack>
    </interceptors>

    <!-- 配置默认的拦截器和拦截器栈引用 -->
    <default-interceptor-ref name="ic"></default-interceptor-ref>

    <!-- 在 Action 中应用默认拦截器和拦截器栈 -->
    <action name="ic" class="example.action.ICAction">
        <result>/success.jsp</result>
    </action>

    <!-- ic1.action 中默认的拦截器和拦截器栈不会起作用 -->
    <action name="ic1" class="example.action.ICAction">
        <result>/success.jsp</result>
        <interceptor-ref name="ic1"></interceptor-ref>
    </action>
</package>
```

在上述配置中，有多个地方配置了拦截器，那么 Struts 2 框架会按如下顺序查找一个 Action 引用了哪些拦截器。

(1) 查找当前 Action 是否声明了拦截器，如果有，则使用这个拦截器，不再继续寻找；如果没有，执行下一步。

(2) 查找当前 Action 所在的 package 中是否使用 default-interceptor-ref 声明了默认

的拦截器引用。如果有，就使用它，不再继续寻找；如果没有，执行下一步。

(3) 递归寻找 Action 所在包的父包中是否设置了默认拦截器引用，直到找到为止。

因此，在代码 7-5 给出的配置中，ic. action 将被应用默认的拦截器栈 ic，而 ic1. action 由于利用 interceptor-ref 进行了重新配置，因此默认拦截器将不发生作用，这也就是为什么我们一再强调，在配置自定义拦截器时不要忘记引用 defaultStack 的原因。

7.4 PreResultListener 接口

PreResultListener 是一个监听器接口，它可以在 Action 完成控制处理之后，Result 执行之前被回调。

PreResultListener 接口只有一个方法。

```
void beforeResult(ActionInvocation invocation,String resultCode)
```

resultCode 是 Action 执行返回的结果码。该方法在 Action 执行之后，Result 执行之前执行。

代码 7-6 所示是添加了 PreResultListener 监听器的 TestInterceptor 拦截器类。

代码 7-6 添加了 PreResultListener 监听器的 TestInterceptor 拦截器类

```
package example.Interceptor;

import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
import com.opensymphony.xwork2.interceptor.PreResultListener;

public class TestInterceptor extends AbstractInterceptor {
    private String icName;
    /* 省略了 get 和 set 方法 */

    public String intercept(ActionInvocation arg0) throws Exception {
        System.out.println(icName+ ":Action 还没有执行!");

        arg0.addPreResultListener(new PreResultListener(){
            public void beforeResult(ActionInvocation arg0, String arg1) {
                System.out.println(icName+ ":beforeResult 方法被调用,
                    Action 已经执行完毕");
            }
        });
        String resultCode= arg0.invoke();
        System.out.println(icName+ ":Action 和 Result 已经执行结束!");
        return resultCode;
    }
}
```

注意： PreResultListener 监听器一定要在 invoke()方法被调用之前注册，否则

beforeResult 将无法被调用。

采用代码 7-4 中的拦截器配置，利用 ic. action 测试上述拦截器，可以得到如图 7-3 所示的输出结果。可以看到，beforeResult()方法在 Action 被执行之后，Result 被执行之前被调用。

```
ic1:Action还没有执行!
正在执行Action.
ic1:beforeResult方法被调用，Action已经执行完毕
ic1:Action和Result已经执行结束!
```

图 7-4　TestInterceptor 运行输出

7.5　案例 4：利用拦截器为留言板增加身份验证功能

通过案例 2 和案例 3，一个基于 Struts 的留言板程序已经具备了雏形。然而，如果一个未登录的用户直接从浏览器访问 listNotes. action 或 detail. action?noteId=1，能够成功查看到所有留言列表以及每条留言的内容。本节将利用拦截器为留言板增加身份验证功能。当用户访问留言板中的页面时，拦截器将截获用户请求，并判断用户是否已经登录。如果没有登录，将强制在浏览器显示登录页面；如果用户已经登录，则允许用户访问请求的资源。

1. 编写身份验证拦截器

AuthenInterceptor 负责判断用户是否已经登录。如果用户已经登录，则将用户请求传递给后续的拦截器、Action 和 Result 的调用；否则，返回 Action. LOGIN 结果码，将请求重定向到登录页面。AuthenInterceptor 类的代码如代码 7-7 所示。代码中给出了详尽的注释，在此不再赘述。

代码 7-7　AuthenInterceptor 类

```
package notes.interceptor;
import java.util.Map;
import com.opensymphony.xwork2.*;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;
public class AuthenInterceptor extends AbstractInterceptor {
    public String intercept(ActionInvocation arg0) throws Exception {

        //获取 ActionContext 上下文,也可以使用 arg0.getInvocationContext()获取
        ActionContext context= ActionContext.getContext();
        /* 获取 session* /
        Map session= context.getSession();
        Object user= session.get("user");
        if(null==user){          /* 未登录,返回 Action.LOGIN 结果码 * /
            //在用户请求访问的 Action 中添加一条 Action 级错误,也可以不加
            ActionSupport action= (ActionSupport)arg0.getAction();
```

```
                action.addActionError("您尚未登录,请先登录!");

                return Action.LOGIN;
            }
            else //将请求传递给后续的拦截器、Action和Result调用
                return arg0.invoke();
        }
    }
```

2 配置Action和拦截器

打开留言板程序的配置文件struts.xml,将其修改成代码7-8所示的形式。

代码7-8 留言板程序的struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="default" namespace="/" extends="struts-default">
        <!--配置安全认证拦截器栈-->
        <interceptors>
            <interceptor name="security"
                class="notes.interceptor.AuthenInterceptor">
            </interceptor>
            <interceptor-stack name="auth">
                <interceptor-ref name="defaultStack"></interceptor-ref>
                <interceptor-ref name="security"></interceptor-ref>
            </interceptor-stack>
        </interceptors>

        <global-results>
            <result name="login">WEB-INF/jsp/login.jsp</result>
        </global-results>

        <!--login不能引用拦截器栈auth-->
        <action name="login" class="notes.action.LoginAction">
            <result type="redirectAction">listNotes.action</result>
        </action>

        <action name="logout" class="notes.action.LoginAction"
            method="logout"/>
        </action>

        <!--让各个Action引用拦截器栈auth-->
        <action name="listNotes" class="notes.action.NotesAction"
            method="list">
            <result name="success">WEB-INF/jsp/index.jsp</result>
            <interceptor-ref name="auth"></interceptor-ref>
```

```
        </action>
        <action name="addNote" class="notes.action.NotesAction"
            method="add">
            <result name="success" type="redirectAction">
                listNotes
            </result>
            <result name="input">WEB-INF/jsp/post.jsp</result>
            <interceptor-ref name="auth"></interceptor-ref>
        </action>
        <action name="detail" class="notes.action.NotesAction"
            method="detail">
            <result name="success">WEB-INF/jsp/detail.jsp</result>
            <interceptor-ref name="auth"></interceptor-ref>
        </action>
    </package>
</struts>
```

注意： 绝对不能为 login.action 配置拦截器 auth，否则任何用户都将无法登录系统。

此时，通过浏览器，在未登录的情况下访问留言板程序中的资源，看是否能够访问成功。

同 步 训 练

为站内短信系统添加身份验证功能，要求只有在用户登录后才能访问系统资源。

Chapter 8
第8章 文件的上传和下载

8.1 文件的上传

8.1.1 文件上传概述

在 Web 应用程序中，经常要用到文件上传功能。例如，在一个 E-mail 系统中，需要提供上传附件的功能；在一个贴图网站中，用户需要利用文件上传将本地的图片贴到网上。

在讨论具体的文件上传技术之前，首先介绍 HTML 的 form 元素的 enctype 属性。该属性用于指定表单数据在发送到服务器之前应该如何编码，可取值包括：

（1）application/x-www-form-urlencoded：表单数据被编码成“名称/值对”的形式，这是默认编码方式。也就是说，在发送到服务器之前，所有字符都会编码（空格转换为“＋”加号，特殊符号转换为 ASCII HEX 值）。

（2）multipart/form-data：表单数据以二进制流的方式传送到服务器。这种编码方式把文件域指定的文件内容也封装到请求参数里。在使用包含文件上传控件的表单时，必须使用该值。

（3）text/plain：表单数据以纯文本形式编码，其中不含任何控件或格式字符。这种方式主要适用于直接通过表单发送邮件。

因此，为了成功上传文件，form 元素的 entype 属性必须设置为 multipart/form-data。另外，form 元素的 method 属性必须设置为 post，这是因为在使用 get 方式提交时，表单中所有的参数都附着在请求 URL 的后面，而 URL 最大长度是有限制的，通常不超过 1KB；而在 post 方式中，表单请求数据封装在 HTTP Header 中，以二进制流的形式传送到 Web 服务器。因此，一个提供文件上传功能的表单，它在浏览器中看到的源码应该具备如下形式：

```
<form action="..." enctype="multipart/form-data" method="post">
    ⋮
    <input type="file" name="..." />
    ⋮
</form>
```

默认情况下，Struts 2 使用 fileUpload 拦截器，利用 Apache 提供的 Common-FileUpload 组件实现文件上传。该组件的优点在于性能优异，支持任意大小文件的上传，

程序员只需要编写少量的代码就能够实现文件的上传操作。

在使用 Common-FileUpload 组件上传文件时，除了需要设置好表单的 enctype 和 method 属性之外，在处理文件上传请求的 Action 类中应该提供三个特殊命名的属性。假设表单中文件选择框(<input type="file" name="xxx" />)的名字为 xxx，那么 Action 类中应该提供如下三个属性。

(1) xxx：上传文件对象，类型为 java.io.File。

(2) xxxFileName：上传文件的名字，类型为 String。

(3) xxxContentType：上传文件的内容类型，类型为 String。

同时，需要为这三个属性定义 get 和 set 方法。

8.1.2 限制上传文件长度和内容类型

fileUpload 拦截器负责完成文件上传工作。这个拦截器已经包含在 defaultStack 拦截器栈中，只要确保配置 Action 时包含了这个栈就可以了。fileUpload 拦截器有两个重要的参数，对文件上传进行控制。

(1) maximumSize：Action 可接受的文件的最大长度(以字节为单位)。默认值为 2MB。

(2) allowedTypes：允许上传内容类型的列表(例如 application/msword)，各类型之间以逗号分隔。当没有指定时，Struts 2 将允许用户上传任意类型的文件。

如果读者对内容类型不熟悉，在 Tomcat 安装目录的 conf 子目录中有一个 web.xml 文件。打开该文件，将看到很多 mime-mapping 元素，例如：

```
<mime-mapping>
    <extension>doc</extension>
    <mime-type>application/msword</mime-type>
</mime-mapping>
```

这里的 extension 元素的值就是平时所说的文件扩展名，mime-type 就是该扩展名的文件类型对应的内容类型，allowedTypes 的值正是由 mime-type 构成的字符串。

如果用户上传的文件大小超过了限定值或者文件类型不匹配，将显示一条错误信息。该错误信息定义在 Struts 2-core-2.x.x.jar 的 struts-message.properties 中，包括如下几个 I18N 键。

```
struts.messages.error.uploading=Error uploading: {0}
                                                    //文件上传的通用错误信息
struts.messages.error.file.too.large=File too large: {0} "{1}" "{2}" {3}
                                                    //上传文件超过了最大长度
struts.messages.error.content.type.not.allowed=Content-Type not al//上传文件的内容类型不允许
```

程序员可以编写自己的错误信息文件，然后在 struts.xml 中利用 constant 标签将文件名赋值给常量 struts.custom.i18n.resources。

需要提醒读者的是，fileUpload 拦截器只是在文件上传到 Web 服务器后判断文件的大小是否超过了 maximumSize 参数的值，如果希望在文件上传到服务器之前就进行判

断，可以使用 Struts 2 常量来设置。Struts 2 中提供了三个与文件上传有关的常量。

(1) struts.multipart.parser：用于指定处理 multipart/form-data 的 MIME 类型(文件上传)请求的框架，可选的值包括 cos、pell 和 jakarta，分别对应使用 cos 的文件上传框架、pell 上传及 common-fileupload 文件上传框架。该属性的默认值为 jakarta。

(2) struts.multipart.saveDir：用于指定上传文件的临时保存路径。该属性的默认值是 javax.servlet.context.tempdir。

(3) struts.multipart.maxSize：用于指定允许上传的文件的最大字节数。默认值为 2MB。

8.1.3 上传单个文件

本节编写一个程序，介绍在 Struts 2 中如何实现文件上传。

1. 编写文件上传页面

新建一个 upload.jsp 文件，内容如代码 8-1 所示。

代码 8-1 文件上传页面(upload.jsp)

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>上传文件</title>
    </head>
    <body>
        <form action="upload" enctype="multipart/form-data" method="post">
            附件:<input type="file" name="file"/>
            <input type="submit" value="提交"/>
        </form>
    </body>
</html>
```

注意：enctype 的值设为 multipart/form-data，method 的值设置为 post。

2. 编写处理文件上传的 Action 类

编写用于处理文件上传的 Action 类，如代码 8-2 所示。

代码 8-2 FileUploadAction.java

```
package example.action;

import java.io.*;
import javax.servlet.ServletContext;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class UploadFileAction extends ActionSupport {
    private File file;                          //代表上传文件的 file 对象
    private String fileFileName;                //上传的文件名
    private String fileContentType;             //上传文件的内容类型
```

```
    //省略了 get 和 set 方法

    public String execute() throws Exception {
        ServletContext context=ServletActionContext.getServletContext();
        String dir=context.getRealPath("/WEB-INF");                    //获取/WEB-INF的真实路径

        BufferedOutputStream bos=null;
        BufferedInputStream bis=null;

        try {
            FileInputStream fis=new FileInputStream(file);
            bis=new BufferedInputStream(fis);
            FileOutputStream fos=new FileOutputStream(new File(dir,
                    fileFileName));
            bos=new BufferedOutputStream(fos);

            byte[] buf=new byte[4096];
            int len=-1;
            while ((len=bis.read(buf)) !=-1) {          //读文件
                bos.write(buf, 0, len);                  //写文件
            }
        } finally {
            try {
                if (null !=bis)
                    bis.close();                         //关闭输入流
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                if (null !=bos)
                    bos.close();                         //关闭输出流
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return SUCCESS;
    }
}
```

提交的文件被上传到 Web 应用程序的 WEB-INF 目录下，新文件名与原始文件名相同。

3. 配置 FileUploadAction

在 struts.xml 中配置 FileUploadAction，如代码 8-3 所示。

代码 8-3　struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"
    "http://struts.apache.org/dtds/struts-2.3.dtd">
<struts>
    <constant name="struts.multipart.maxSize" value="10485760"></constant>
    <package name="default" namespace="/" extends="struts-default">
        <action name="upload" class="example.action.FileUploadAction">
            <result>/success.jsp</result>
            <interceptor-ref name="defaultStack">
                <param name="fileUpload.maximumSize">10485760</param>
                <param name="fileUpload.allowedTypes">
                    image/jpeg,image/gif,application/x-rar-compressed
                </param>
            </interceptor-ref>
        </action>
    </package>
</struts>
```

上述配置将 Struts 2 允许上传的最大文件长度和 Action 允许处理的最大长度设置为 10MB，并且只允许上传扩展名为.jpg、.gif 和.rar 的文件。

8.1.4 上传多个文件

如果一次希望上传多个文件，需要在 form 表单中使用多个 name 相同的 file 标签（或者 HTML 中的<input type="file" ... />），例如：

```
<s:form action="upload2" enctype="multipart/form-data">
    <s:file name="file" label="附件"></s:file>
    <s:file name="file" label="附件"></s:file>
    <s:file name="file" label="附件"></s:file>
    <s:submit value="提交"/>
</s:form>
```

在接收文件上传处理的 Action 类中，可以使用数组或 List 接收上传文件的信息。当使用数组接收时，Action 类中需要提供以下三个属性及其 get 和 set 方法。

```
private File[] file;
private String[] fileFileName;
private String[] fileContentType;
```

当使用 List 接收时，Action 类中需要提供以下三个属性及其 get 和 set 方法。

```
private List<File> file;
private List<String> fileFileName;
private List<String> fileContentType;
```

代码 8-4 给出了一个用于处理上述表单上传多文件的 Action 类，该类利用 List 接收上传文件的信息。

代码 8-4　用 List 实现多文件上传

```
package example.action;

import java.io.* ;
import java.util.List;
import javax.servlet.ServletContext;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class FileUploadAction2 extends ActionSupport {
    private List<File> file;                    //代表上传文件的 file 对象列表
    private List<String> fileFileName;          //上传文件的文件名列表
    private List<String> fileContentType;       //上传文件的内容类型列表
    //省略了 get 和 set 方法

    public String execute() throws Exception {
        ServletContext context=ServletActionContext.getServletContext();
        String dir=context.getRealPath("/WEB-INF");              //获取 /WEB-INF 的真实路径
        BufferedOutputStream bos=null;
        BufferedInputStream bis=null;

        for (int i=0; i<file.size(); i++) {     //逐个处理 List 中的每个文件
        try {
            FileInputStream fis=new FileInputStream(file.get(i));
            bis=new BufferedInputStream(fis);   //定义输入流
            FileOutputStream fos=new FileOutputStream(new File(dir,
                        fileFileName.get(i)));
            bos=new BufferedOutputStream(fos);  //定义输出流
            byte[] buf=new byte[4096];          //定义读写缓冲区
            int len=-1;
            while ((len=bis.read(buf)) !=-1) {  //读文件
                bos.write(buf, 0, len);         //写文件
            }
        } finally {
            try {
                if (null !=bis)
                    bis.close();                //关闭输入流
            } catch (IOException e) {
                e.printStackTrace();
            }
            try {
                if (null !=bos)
                    bos.close();                //关闭输出流
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    return SUCCESS;
```

```
    }
}
```

利用数组接收多文件的方法与上述代码大同小异，读者可以自行练习一下。

8.2 文件的下载

8.2.1 文件下载概述

在研究了文件的上传之后，下面介绍文件的下载。有的读者可能会想，只要为每个文件提供一个链接，不就可以下载了吗？这种方法的确能够实现文件的下载，但是会引发几个问题。

(1) 由于暴露了文件的真实地址，一些没有获得授权的用户也能够下载该文件。

(2) 对于一些珍贵的、独家提供下载的资源，无法防止其他网站跨站点引用它们。

(3) 服务器端文件只能存放在 Web 应用程序所在的目录下(WEB-INF 子目录除外)。

事实上，有时为了保护文件不被非法下载或链接，可能会把它们存放在 WEB-INF 子目录下、Web 应用程序安装目录以外的位置或是数据库中，此时不能通过一个静态的超级链接提供下载，必须通过编程的形式动态生成下载地址。

8.2.2 stream结果类型

Struts 2 专门提供了一种 stream 结果类型来支持文件下载。stream 结果类型具有 7 个可选参数，如表 8-1 所示。

表 8-1 stream 结果类型的参数

参数名称	默认值	说明
contentType	text/plain	下载文件的内容类型
contentLength		下载文件的长度，供浏览器显示进度条
contentDisposition	inline	指定文件下载的默认名字。如果不指定，则使用 Action 名.action
inputName	inputStream	Action 中用于返回 InputStream 的 get 方法的名字，类型为 InputStream
bufferSize	1KB	文件读写缓冲区的大小，以字节为单位
allowCaching	true	是否允许浏览器缓存
contentCharSet		HTTP 响应头信息中的编码方式

参数 contentDisposition 的可选值包括以下两个。

(1) inline;filename="下载文件名"：inline 表示在浏览器中打开该文件。inline 可以省略。

(2) attachment;filename="下载文件名"：attachment 表示弹出“文件下载”对话框。

当然，如果浏览器不支持该文件类型的显示，无论指定的是 inline 还是 attachment，都会弹出“文件下载”对话框。

下面给出一个 stream 结果类型的例子。

```
<result type="stream">
    <param name="contentType">image/jpeg</param>
    <param name="inputName">imageStream</param>
    <param name="contentDisposition">attachment;filename="photo.jpg"</param>
    <param name="bufferSize">1024</param>
</result>
```

8.2.3　文件下载实例

下面给出一个文件下载的例子。

1. 编写文件下载页面

编写文件下载页面，如代码 8-5 所示。

代码 8-5　文件下载页面

```
<%@ page language="java" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>下载文件</title>
    </head>
    <body>
        <a href="download.action">下载文件</a>
    </body>
</html>
```

2. 编写处理文件下载的 Action 类

新建一个 FileDownloadAction 类，用于处理文件下载，如代码 8-6 所示。

代码 8-6　FileDownloadAction.java

```
package example.action;

import java.io.*;
import com.opensymphony.xwork2.ActionSupport;

public class FileDownloadAction extends ActionSupport {

    //创建 inputStream,为 stream 流的 InputName 参数提供值
    public InputStream getInputStream() throws Exception {

        File file=new File("d:\photo\IMG_2527.jpg");
            return new FileInputStream(file);
    }

    public String execute() throws Exception {
        return SUCCESS;
```

```
        }

        //提供转换编码后的供下载用的文件名
        public String getDownloadFileName() {
            String downFileName= "照片.jpg"; //下载到客户端的文件名
            try {
                //解决中文文件名乱码问题
                downFileName= new String(downFileName.getBytes(), "ISO8858-1");
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
            return downFileName;
        }
    }
```

FileDownloadAction 类提供了为 stream 的 InputName 参数的默认值 inputStream 提供了 get 方法。

3. 配置 FileDownloadAction

编辑 struts.xml,并配置 FileDownloadAction,如代码 8-7 所示。

代码 8-7 在 struts.xml 中配置 FileDownloadAction

```
<action name= "download" class= "example.action.FileDownloadAction">
    <result name= "success" type= "stream">
        <param name= "contentDisposition">
            attachment;filename= "${downloadFileName}"
        </param>
        <param name= "bufferSize">4096</param>
    </result>
</action>
```

上述代码中的${downloadFileName}将调用 FileDownloadAction 的 getDownloadFileName 方法获取保存到客户端的文件名。

在本例子中,待下载的文件名被硬编码在 FileDownloadAction 类中。在实际的应用中,肯定不会采用这种方法。在本章的案例部分,将研究任意文件的下载。

在调试文件下载 Action 时,经常会看到下面的错误信息。

```
Can not find a java.io.InputStream with the name [inputStream] in the invocation stack. Check the <
param name= "inputName"> tag specified for this action.
```

产生这种错误的原因是因为 InputStream 没有创建成功,取值为 null。读者可以向控制台输出待下载文件的真实路径,看是否存在错误。

8.3 案例 5: 为留言板程序添加附件功能

8.3.1 为留言板添加上传附件功能

如果希望用户在留言时可以上传附件,需要为留言板添加文件上传功能。

1. 修改留言页面 post.jsp

修改留言页面 post.jsp，增加 file 标签，如代码 8-8 所示（为节约篇幅，仅给出表单内容）。

代码 8-8　添加了 file 标签的留言页面

```
<s:form action="addNote.action" theme="simple"
        enctype="multipart/form-data" method="post">
    <div class="col1" style="text-align: right">标题</div>
    <div class="col2">
        <s:textfield name="note.title" size="50"></s:textfield>
    </div>
    <div class="clear"></div>
        <div class="col1" style="text-align: right">内容</div>
        <div class="col2">
            <s:textarea rows="20" cols="60" name="note.content">
            </s:textarea>(少于 1000 字)
        </div>
        <div class="clear"></div>
        <div class="col1" style="text-align: right">附件</div>
        <div class="col2">
            <s:file name="attachment"></s:file>
        </div>
        <div class="col2">
            <s:submit value="确定"></s:submit>
            <s:reset value="重填"></s:reset>
        </div>
</s:form>
```

由于 form 标签使用了 simple 主题，因此必须显式地将 method 属性设置为 post。

2 修改处理留言的 Action 类

修改处理留言的 NotesAction 类，为该类添加接收上传文件信息的三个属性，并编写文件上传代码，如代码 8-9 所示。

代码 8-9　NotesAction.java

```
package notes.action;

import java.io.*;
import java.util.*;
import notes.dao.NotesDao;
import notes.dao.impl.NotesDaoImpl;
import notes.model.*;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.*;

public class NotesAction extends ActionSupport {
    private Notes note;
```

```
private List<Notes> notes=new ArrayList<Notes>();
private String uploadDir;          //上传文件的相对路径,从 struts.xml 中获得
private File attachment;                    //代表上传文件的 file 对象
private String attachmentFileName;          //上传文件的文件名
private String attachmentContentType;       //上传文件的内容类型
//省略了 get 和 set 方法

public String list() {                      //列出所有的留言
    NotesDao notesDao=new NotesDaoImpl();
    notes=notesDao.getAllNotes();
    return SUCCESS;
}

public String add() {                       //添加新留言
    if (null==note)
        return INPUT;

    ActionContext context=ActionContext.getContext();
    Map session=context.getSession();
    User user=(User) session.get("user");
    if (null==user)
        return LOGIN;

    if(null!=attachment)                    //如果当前的留言有附件,上传
        upload();

    note.setUser(user);
    NotesDao notesDao=new NotesDaoImpl();
    notesDao.addNote(note);
    return SUCCESS;
}

public String detail() {                    //取得某个留言的详细信息
    NotesDao notesDao=new NotesDaoImpl();
    note=notesDao.getNoteById(noteId);
    return SUCCESS;
}
/*完成文件上传操作*/
private void upload(){
    //获得存储附件的目录的真实路径
    String path=ServletActionContext.getServletContext()
            .getRealPath(uploadDir);
    File dir=new File(path);
    //存放附件的目录如果不存在,则创建
    if(!dir.exists())
        dir.mkdir();

    /*得到当前时间自 1970年 1月 1日 0时 0分 0秒开始流逝的毫秒数,用其作为新文件的
     文件名*/
```

```
String newFileName;
long now=new Date().getTime();
int index=attachmentFileName.lastIndexOf('.');
//判断文件是否有扩展名
if(index!=-1)
    newFileName=now+this.attachmentFileName.substring(index);
else
    newFileName=Long.toString(now);

byte[] buf=new byte[4096];
BufferedInputStream bis=null;
BufferedOutputStream bos=null;

//读取保存在临时目录下的上传文件,写入新文件
try{
    FileInputStream fis=new FileInputStream(attachment);
    bis=new BufferedInputStream(fis);
    FileOutputStream fos=new FileOutputStream(
            new File(dir,newFileName));
    bos=new BufferedOutputStream(fos);

    int len=-1;
    while((len=bis.read(buf))!=-1)
        bos.write(buf,0,len);

    //在留言内容后面添加一条下载附件的链接
    String content=note.getContent();
    content+="</br>";
    content+="<a href='";
    content+="download.action?fileName="+newFileName;
    content+="'>附件</a>";

    note.setContent(content);
    note.setAttachment(newFileName);
}catch(IOException e){
    e.printStackTrace();
}
finally{

    try{
    if(null!=bis)
        bis.close();

    if(null!=bos)
        bos.close();
    }catch(IOException e){
        e.printStackTrace();
    }
}
```

```
        }
    }
```

NotesAction 类在上传文件时，构造了一个利用 download.action 下载附件的超级链接，并将其附加在留言内容的末尾，用户可以通过点击该超级链接下载附件。另外，在上传文件时，为了解决文件重名问题，利用自 1970 年 1 月 1 日 0 时 0 分 0 秒开始流逝的毫秒数作为新文件的文件名。

属性 uploadDir 存放上传文件的目录名，可以在 struts.xml 中配置 NotesAction 时指定，以利于系统维护。关于 NotesAction 类在 struts.xml 中的配置，将在解决完文件下载之后一起讲解。

8.3.2 为留言板添加下载附件功能

1. 文件下载 Action 类

在上传文件时，下载附件的超级链接已经添加在留言内容的末尾，形如：

```
download.action?fileName=1344844497781.jpg
```

在编写处理下载文件的 Action 类时，只需通过 fileName 属性获取下载的文件名，然后构造一个 InputStream 对象就可以了，如代码 8-10 所示。

代码 8-10 文件下载类

```
package notes.action;
import java.io.*;
import javax.servlet.ServletContext;
import org.apache.struts2.ServletActionContext;
import com.opensymphony.xwork2.ActionSupport;
public class FileDownloadAction extends ActionSupport {
    private String fileName;            //初始的通过 param 指定的文件名属性
    private String downloadDir;         //下载文件所在的目录

    public void setFileName(String fileName) {
        this.fileName=fileName;
    }
    public void setDownloadDir(String downloadDir) {
        this.downloadDir=downloadDir;
    }

    public InputStream getInputStream() throws Exception {
        //通过 ServletContext,也就是 application 来读取数据
        ServletContext context=ServletActionContext.getServletContext();
        String path=context.getRealPath(downloadDir);

        String downloadFile=path+"\\"+fileName;

        //构造关于下载文件的 InputStream 对象
        File file=new File(downloadFile);
            return new FileInputStream(file);
    }
```

```
        public String execute() throws Exception {
            return SUCCESS;
        }
        /*提供转换编码后的供下载用的文件名*/
        public String getDownloadFileName() {
            String downFileName= fileName;
            try {
                downFileName= new String(downFileName.getBytes(), "ISO8858-1");
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
            return downFileName;
        }
    }
```

2 修改 struts.xml

修改留言板程序的 struts.xml，添加 FileDownloadAction 的配置，并重新配置 NotesAction 类，如代码 8-11 所示。

代码 8-11　struts.xml

```
...
<package name="default" namespace="/" extends="struts-default">
...
        <action name="addNote" class="notes.action.NotesAction"
            method="add">
            <result type="redirectAction">listNotes</result>
            <result name="input">WEB-INF/jsp/post.jsp</result>
            <param name="uploadDir">uploadFiles</param>
            <interceptor-ref name="auth"></interceptor-ref>
        </action>
        <action name="download" class="notes.action.FileDownloadAction">
            <param name="downloadDir">uploadFiles</param>
            <result name="success" type="stream">
                <param name="contentDisposition">
                    attachment;filename="${downloadFileName}"
                </param>
                <param name="bufferSize">4096</param>
            </result>
        </action>
</package>
...
```

在配置 NotesAction 的 addNotes 方法时，通过一个静态参数为 NotesAction 实例的 uploadDir 属性赋值，用于设置上传文件的路径。注意，由代码 8-9 可知，上传文件只能存放在 Web 应用程序部署目录的某个子目录中。另外，在配置 FileDownloadAction 时，通过一个静态参数为该类实例的 downloadDir 属性赋值，其值应该和 uploadDir 属性的值一致。

同步训练

为站内短信系统添加附件上传和下载功能。

方法 1：附件下载链接可以附着在短信内容之后。

方法 2：将附件的地址单独存放在数据表的一个字段中，在下载页面提供一个单独的下载链接。

Chapter 9

第9章 输入验证

9.1 输入验证概述

一个 Web 应用程序必须保证用户输入的数据是有效的。例如，在一个注册程序中，用户名是否填写，输入的口令长度是否合法，两次口令是否一致，输入的 E-mail 地址是否满足格式要求等。一个健壮的应用程序不会因为用户输入的数据不合法而显示一大堆错误信息，也不会因为恶意用户输入的伪造数据而导致系统信息泄露。

在 Web 应用程序开发中，常用的数据验证技术包括客户端验证和服务器验证两种。客户端验证主要是通过 JavaScript 脚本实现的，验证速度快，若有不符合要求的输入，响应信息能够快速地返回给用户。由于验证数据不需要提交给服务器，客户端验证不会加重服务器的负载。但是，客户端验证存在一个致命缺点。目前，很多工具可以在输入数据经过客户端验证后，浏览器发送请求前，截取数据，恶意用户可以修改请求中的数据，将恶意数据注入服务器。服务器端的验证可以弥补客户端验证的不足，它除了能够实现客户端验证提供的功能外，还能够实现数据逻辑的验证，例如验证码是否正确等。

Struts 2 提供了一套验证框架，能够完成输入数据的常规验证工作。使用验证框架，程序员只要在一个格式为 XML 的验证文件中声明输入数据应该满足的规则，例如哪些字段需要验证，应该满足什么验证条件，当验证失败时应该把什么样的出错消息发送到客户端，等等。

Struts 2 验证框架中内置了很多验证器，每个验证器对应一个 Java 类。当用户提交输入数据时，Validator 拦截器负责调用验证器验证输入数据是否满足规则。Validator 拦截器已经包含在 defaultStack 拦截器栈中，只要在配置 struts. xml 时确保 Action 类引用了 defaultStack，就无须手工引用 Validator 拦截器了。

利用 Struts 验证器进行输入验证的步骤如下。

(1) 确定需要验证的 Action 类。

(2) 为每个待验证的 Action 类编写验证配置文件。验证配置文件应该和 Action 类放在同一个 Java 包中，文件名应该是以下两种格式中的一种。

```
ClassName-validation.xml
ClassName-alias-validation.xml
```

其中，ClassName 表示待验证的 Action 类的名字；alias 表示在 struts. xml 中配置的 Action 的名字。

当 Action 类只有一个动作，或者虽然有多个动作，但是这些动作的验证规则一致时，可以使用第一种格式为验证配置文件命名。如果一个 Action 类中的不同动作的验证规则不一致，必须为每个动作分别编写验证配置文件，这时候采用第二种格式。当一个 Action 类同时具有以上两种格式的验证配置文件时，例如 RegisterAction-validation. xml 和 RegisterAction-reg-validation. xml，当用户访问/reg. action 时，验证框架将首先加载 RegisterAction-validation. xml，然后加载 RegisterAction-reg-validation. xml，并且两个验证配置文件的验证规则是叠加的，而不是覆盖的。

9.2 验证配置文件的结构

http://www. opensymphony. com/xwork/xwork-validator-1. 0. 3. dtd 定义了验证配置文件的结构，这个文件在 xwork-core-x. x. x. jar 中也可以找到。

xwork-validator-1. 0. 3. dtd 定义了两种类型的验证器。

(1) 字段验证器(Field Validator)：字段验证器与表单中的某个输入标签相关联，在将输入标签的值赋给一个 Action 动作的属性之前会发生验证行为。Struts 2 内置的验证器基本上都是字段验证器。

(2) 普通验证器(Non-Field Validator)：普通验证器不和表单中某个特定的输入标签相关联，它们主要用来测试几个输入标签之间是否满足某种特定的条件。

字段验证器在验证失败后，会将错误消息放到 Action 类字段错误消息中，可以在页面中利用 Struts 的 fielderror 标签显示错误信息。普通验证器在验证失败后，会将消息放到 Action 级别的错误消息中，可以在页面中利用 Struts 的 actionerror 标签显示错误信息。

代码 9-1 给出了 xwork-validator-1. 0. 3. dtd 的内容。

代码 9-1　xwork-validator-1. 0. 3. dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    XWork Validators DTD.
    Used the following DOCTYPE.
    <!DOCTYPE validators PUBLIC
        "-//Apache Struts//XWork Validator 1.0.3//EN"
        "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">
-->
<!ELEMENT validators (field|validator)+>
<!ELEMENT field (field-validator+)>
<!ATTLIST field
    name CDATA #REQUIRED
>
<!ELEMENT field-validator (param*, message)>
<!ATTLIST field-validator
    type CDATA #REQUIRED
        short-circuit (true|false) "false"
>
<!ELEMENT validator (param*, message)>
```

```
<!ATTLIST validator
    type CDATA #REQUIRED
        short-circuit (true|false) "false"
>
<!ELEMENT param (#PCDATA)>
<!ATTLIST param
    name CDATA #REQUIRED
>
<!ELEMENT message (#PCDATA|param)* >
<!ATTLIST message
    key CDATA #IMPLIED
>
```

从 DTD 文件中可以看到验证配置文件的 DOCTYPE 和包含的各个元素。下面简单介绍各个元素的用法。

(1) validators 元素：validators 是验证配置文件的根元素，它可以有任意个数的 field 和 validator 子元素。

(2) field 元素：每个 field 元素对应 Action 类的一个属性，该属性的值需要被一个或多个字段验证器验证。field 元素有一个必需的属性 name，必须和表单标签的 name 属性取值一致。在 field 元素中应该包含一个以上 field-validator 子元素，用于指明要使用的验证器的类型。

(3) field-validator 元素：用于配置对 Action 属性的验证规则。它包含一个必需的属性 type，用于指定要使用的验证器的类型。field-validator 元素还包含一个可选的属性 short-ciruit，用于指示在本验证器失败时是否阻止其他验证器的执行，默认值为 false。field-validator 元素的内部必须包含一个 message 子元素，还可以包括 0 个或多个 param 子元素。

(4) message 元素：用于指定验证失败时的错误消息，它有一个可选属性 key，用于读取国际化资源文件中的本地消息。

(5) param 元素：用于向验证器传递参数值。它包含一个必需的 name 属性，用于指示参数的名称。

(6) validator 元素：用于声明字段验证器或者普通验证器。它包含一个必需的属性 type，用于指定要使用的验证器的类型。与 field 元素一样，validator 元素也包含一个可选的属性 short-ciruit，用于指示在本验证器失败时是否阻止其他验证器的执行，默认值为 false。field-validator 元素也必须包含一个 message 子元素，还可以包括 0 个或多个 param 子元素。如果使用 validator 元素声明一个字段验证器，必须定义一个名为 fieldName 的参数来指示所要验证的表单标签。

9.3 Struts 2 内置的验证器

xwork-core-2.3.4.jar 提供了 16 种内置的验证器，包括 required 验证器、requiredstring 验证器、int 验证器、long 验证器、short 验证器、double 验证器、date 验证器、expression

验证器、fieldexpression 验证器、email 验证器、url 验证器、visitor 验证器、conversion 验证器、stringlength 验证器、regex 验证器和 conditionalvisitor 验证器等。

9.3.1 required 验证器

required 验证器用于检查指定的 Action 属性的值是否不为 null。表 9-1 给出了 required 验证器的参数。

表 9-1 required 验证器的参数

参数名称	数据类型	说　明
fieldname	String	指示所要验证的 Action 属性名。当使用 validator 元素验证时，需要提供该参数；使用 field 元素验证时，不需要提供该参数

代码 9-2 所示为 required 验证器的一个示例。

代码 9-2 required 验证器示例

```
<validators>
    <!-- validator 声明 required 验证器,属性 type 指示了验证器的类型,参数 fieldName 指示了所
     要验证的 Action 属性,子元素 message 给出了验证失败时的错误信息 -->
    <validator type="required">
        <param name="fieldName">title</param>
        <message>留言主题不能为空值</message>
    </validator>

    <!-- field 声明 required 验证器,属性 name 指示了所要验证的 Action 属性,子元素 field-
    validator 的属性 type 指示了验证器的类型,子元素 message 给出了验证失败时的错误信息 -
    ->
    <field name="title">
        <field-validator type="required">
            <message>留言主题不能为空值</message>
        </field-validator>
    </field>
</validators>
```

9.3.2 requiredstring 验证器

requiredstring 验证器用于检查指定的 Action 属性值不为 null，并且长度应该大于 0（也就是 Action 属性值不能为""）。requiredstring 验证器的参数如表 9-2 所示。

表 9-2 requiredstring 验证器的参数

参数名称	数据类型	说　明
fieldname	String	指示所要验证的 Action 属性名。当使用 validator 元素验证时，需要提供该参数；使用 field 元素验证时，不需要提供该参数
trim	Boolean	在验证前是否要删除属性首尾的空格，默认值为 true

代码 9-3 所示为 requiredstring 验证器的一个示例。

代码 9-3　requiredstring 验证器示例

```
<validators>
<!--validator 声明 requiredstring 验证器。参数 trim 的默认值为 true,可以省略。-->
    <validator type="requiredstring">
        <param name="fieldName">title</param>
        <param name="trim">true</param>
        <message>留言主题不能为空值</message>
    </validator>

<!--field 声明 requiredstring 验证器。参数 trim 的默认值为 true,可以省略。-->
    <field name="title">
        <field-validator type="requiredstring">
            <param name="trim">true</param>
            <message>留言主题不能为空值</message>
        </field-validator>
    </field>
</validators>
```

9.3.3　int、long 和 short 验证器

int 验证器、long 验证器和 short 验证器分别用于检查指定的 Int 型整数、Long 型整数和 Short 型整数是否在某个范围内。三个验证器具有相同的参数,如表 9-3 所示。

表 9-3　int 验证器、long 验证器和 short 验证器的参数

参数名称	数据类型	说　明
fieldname	String	指示所要验证的 Action 属性名。当使用 validator 元素验证时,需要提供该参数;使用 field 元素验证时,不需要提供该参数
max	Int/Long/Short	最大值。如果没有指定该参数,则没有最大值限制
min	Int/Long/Short	最小值。如果没有指定该参数,则没有最小值限制

代码 9-4 所示为 int 验证器的一个示例。long 验证器和 short 验证器的用法与 int 型一致,不再额外举例。

代码 9-4　int 验证器示例

```
<validators>
    <!--validator 声明 int 验证器。参数 min 指示 age 的最小值为 18,参数 max 指示 age 的最大值
     为 45-->
    <validator type="int">
        <param name="fieldName">age</param>
        <param name="min">18</param>
        <param name="max">45</param>
        <message>年龄必须在 ${min}和 ${max}之间</message>
    </validator>

    <!--field 声明 int 验证器。参数 min 指示 age 的最小值为 18,参数 max 指示 age 的最大值为 45
     -->
```

```
        <field name="age">
            <field-validator type="int">
                <param name="min">18</param>
                <param name="max">45</param>
                <message>年龄必须在 ${min}和 ${max}之间</message>
            </field-validator>
        </field>
    </validators>
```

上述代码的 message 元素中出现了两个 OGNL 表达式 ${min}和 ${max}。${min}表示取参数 min 的值，${max}表示取参数 max 的值。

9.3.4 double验证器

double 验证器用于检查指定的双精度浮点数是否在某个范围内。double 验证器的参数如表 9-4 所示。

表 9-4 double 验证器的参数

参数名称	数据类型	说　　明
fieldname	String	指示所要验证的 Action 属性名。当使用 validator 元素验证时，需要提供该参数；使用 field 元素验证时，不需要提供该参数
minInclusive	Double	指定双精度浮点数的最小值，待验证的值可以等于该值。如果没有指定该参数，则没有最小值限制
maxInclusive	Double	指定双精度浮点数的最大值，待验证的值可以等于该值。如果没有指定该参数，则没有最大值限制
minExclusive	Double	指定双精度浮点数的最小值，待验证的值必须大于该值。如果没有指定该参数，则没有最小值限制
maxExclusive	Double	指定双精度浮点数的最大值，待验证的值必须小于该值。如果没有指定该参数，则没有最大值限制

代码 9-5 所示为 double 验证器的一个示例。

代码 9-5 double 验证器示例

```
    <validators>
        <!-- validator声明 double验证器 -->
        <validator type="double">
            <param name="fieldName">salary</param>
            <param name="minInclusive">1000.0</param>
            <param name="maxInclusive">9999.99</param>
            <message>
                工资必须大于等于${minInclusive}并且小于等于${maxInclusive}
            </message>
        </validator>

        <!-- field声明 double验证器 -->
        <field name="percentage">
            <field-validator type="double">
```

```
            <param name="minExclusive">999.99</param>
            <param name="maxExclusive">10000.0</param>
            <message>
                工资必须大于${minExclusive}并且小于 ${maxExclusive}
            </message>
        </field-validator>
    </field>
</validators>
```

9.3.5 date 验证器

date 验证器用于检查指定的日期型数据是否在某个范围内。表 9-5 给出了 date 验证器的参数。

表 9-5　date 验证器的参数

参数名称	数据类型	说　　明
fieldname	String	指示所要验证的 Action 属性名。当使用 validator 元素验证时，需要提供该参数；使用 field 元素验证时，不需要提供该参数
min	Double	指定日期的最小值。如果没有指定该参数，则没有最小值限制
max	Double	指定日期的最大值。如果没有指定该参数，则没有最大值限制

代码 9-6 所示为 date 验证器的一个示例。

代码 9-6　date 验证器示例

```
<validators>
    <!-- validator声明 date验证器 -->
    <validator type="date">
        <param name="fieldName">birthday</param>
        <param name="min">01/01/1990</param>
        <param name="max">08/01/2012</param>
        <message>出生日期应该在 ${min}和 ${max}之间</message>
    </validator>

    <!-- field声明 date验证器 -->
    <field name="birthday">
        <field-validator type="date">
            <param name="min">01/01/1990</param>
            <param name="max">08/01/2012</param>
            <message>出生日期应该在 ${min}和 ${max}之间</message>
        </field-validator>
    </field>
</validators>
```

9.3.6 expression 验证器

expression 验证器用于检查给定的 OGNL 表达式的值是否为真。该验证器是一个

普通验证器，只能使用 validator 元素验证。表 9-6 给出了 expression 验证器的参数。

表 9-6　expression 验证器的参数

参数名称	数据类型	说　明
expression	Boolean	指定要计算的 OGNL 表达式，该表达式基于 Value Stack 求值。如果计算结果为真，则通过验证，否则验证失败

利用 expression 验证器验证 Value Stack 中的 password 和 repassword 的值是否一致，如代码 9-7 所示。

代码 9-7　expression 验证器示例

```
<!-- validator 声明 expression 验证器，验证 repassword 的值必须与 password 一致 -->
<validators>
    <validator type="expression">
        <param name="expression">password==repassword</param>
        <message>两次密码输入不一致!</message>
    </validator>
</validators>
```

9.3.7　fieldexpression 验证器

fieldexpression 验证器用于检查给定的 OGNL 表达式的值是否为真。它的作用与 expression 验证器类似，只不过 fieldexpression 验证器既可以利用 validator 元素验证，也可以使用 field 元素验证。表 9-7 给出了 fieldexpression 验证器的参数。

表 9-7　fieldexpression 验证器的参数

参数名称	数据类型	说　明
fieldname	String	指示所要验证的 Action 属性名。当使用 validator 元素验证时，需要提供该参数；使用 field 元素验证时，不需要提供该参数
expression	Boolean	指定要计算的 OGNL 表达式，该表达式基于 Value Stack 求值。如果计算结果为真，则通过验证，否则验证失败

代码 9-8 所示为 fieldexpression 验证器的一个示例。

代码 9-8　fieldexpression 验证器示例

```
<validators>
    <!-- validator 声明 fieldexpression 验证器，验证 repassword 的值必须与 password 一致 -->
    <validator type="fieldexpression">
        <param name="fieldName">repassword</param>
        <param name="expression">password==repassword</param>
        <message>两次密码输入不一致!</message>
    </validator>

    <!-- field 声明 fieldexpression 验证器，验证 repassword 的值必须与 password 一致 -->
    <field name="repassword">
        <field-validator type="fieldexpression">
```

```
            <param name="expression">password==repassword</param>
            <message>两次密码输入不一致!</message>
        </field-validator>
    </field>
</validators>
```

9.3.8　regex 验证器

regex 验证器使用正则表达式验证给定的字符串是否满足某种格式。表 9-8 给出了 regex 验证器的参数。

表 9-8　regex 验证器的参数

参数名称	数据类型	说　　明
fieldname	String	指示所要验证的 Action 属性名。当使用 validator 元素验证时，需要提供该参数；使用 field 元素验证时，不需要提供该参数
expression	Boolean	指定要计算的 OGNL 表达式，该表达式基于 Value Stack 求值。如果计算结果为真，则通过验证，否则验证失败

代码 9-9 所示为 regex 验证器的一个示例。

代码 9-9　regex 验证器示例

```
<validators>
    <!-- validator 声明正则验证器,验证 password 属性 -->
    <validator type="regex">
        <param name="fieldName">password</param>
        <!--指定匹配的正则表达式-->
        <param name="expression"><![CDATA[(\w{6,20})]]></param>
        <message>密码长度必须在 6 到 20 之间,且必须是字母或者数字</message>
    </validator>

    <!-- field 声明正则验证器,验证 password 属性 -->
    <field name="password">
        <field-validator type="regex">
            <!--指定匹配的正则表达式-->
            <param name="expression"><![CDATA[(\w{6,20})]]></param>
            <message>密码长度必须在 6 到 20 之间,且必须是字母或者数字</message>
        </field-validator>
    </field>
</validators>
```

9.3.9　email 验证器

email 验证器扩展自 regex 验证器，利用正则表达式验证给定的字符串是否是合法的 E-mail 地址。email 验证器用到的正则表达式为：

```
\\b(^['_A-Za-z0-9-]+(\\.['_A-Za-z0-9-]+)*@([A-Za-z0-9-])+(\\.[A-Za-z0-
```

```
9-]+)*((\\.[A-Za-z0-9]{2,})|(\\.[A-Za-z0-9]{2,}\\.[A-Za-z0-9]{2,}))$)\\b
```

表9-9给出了email验证器的参数。

表9-9 email验证器的参数

参数名称	数据类型	说明
fieldname	String	指示所要验证的Action属性名。当使用validator元素验证时，需要提供该参数；使用field元素验证时，不需要提供该参数

代码9-10所示为email验证器的一个示例。

代码9-10 email验证器示例

```
<validators>
    <!-- validator声明 email验证器 -->
    <validator type="email">
        <param name="fieldName">myEmail</param>
        <message>Email地址不合法</message>
    </validator>

    <!-- field声明 email验证器 -->
    <field name="myEmail">
        <field-validator type="email">
            <message>Email地址不合法</message>
        </field-validator>
    </field>
</validators>
```

9.3.10 url验证器

url验证器用于验证给定的字符串是否是合法的URL。表9-10给出了url验证器的参数。

表9-10 url验证器的参数

参数名称	数据类型	说明
fieldname	String	指示所要验证的Action属性名。当使用validator元素验证时，需要提供该参数；使用field元素验证时，不需要提供该参数

代码9-11所示为url验证器的一个示例。

代码9-11 url验证器示例

```
<validators>
    <!-- validator声明 url验证器 -->
    <validator type="url">
        <param name="fieldName">myURL</param>
        <message>给定的URL不合法</message>
    </validator>

    <!-- field声明 url验证器 -->
    <field name="myURL">
        <message>给定的URL不合法</message>
```

```
    </field>
</validators>
```

9.3.11　conversion 验证器

conversion 验证器用于验证指定的字段在类型转换过程中是否发生错误。表 9-11 给出了 conversion 验证器的参数。

表 9-11　conversion 验证器的参数

参数名称	数据类型	说　　明
fieldname	String	指示所要验证的 Action 属性名。当使用 validator 元素验证时，需要提供该参数；使用 field 元素验证时，不需要提供该参数

代码 9-12 所示为 conversion 验证器的一个示例。

代码 9-12　conversion 验证器示例

```
<validators>
    <!-- validator 声明 conversion 验证器 -->
    <validator type="conversion">
        <param name="fieldName">age</param>
        <message>类型转换错误，请输入一个整数</message>
    </validator>

    <!-- field 声明 conversion 验证器 -->
    <field name="age">
        <field-validator type="conversion">
            <message>类型转换错误，请输入一个整数</message>
        </field-validator>
    </field>
</validators>
```

9.3.12　stringlength 验证器

stringlength 验证器用于验证指定的字段在类型转换过程中是否发生错误。表 9-12 给出了 stringlength 验证器的参数。

表 9-12　stringlength 验证器的参数

参数名称	数据类型	说　　明
fieldname	String	指示所要验证的 Action 属性名。当使用 validator 元素验证时，需要提供该参数；使用 field 元素验证时，不需要提供该参数
minLength	Double	指定字符串的最小值。如果没有指定该参数，则没有最小值限制
maxLength	Double	指定字符串的最大值。如果没有指定该参数，则没有最大值限制
trim	Boolean	指定在计算长度前是否去掉字符串首、尾的空格，默认值为 true

代码 9-13 所示为 stringlength 验证器的一个示例。

代码 9-13 stringlength 验证器示例

```
<validators>
    <!-- validator 声明 stringlength 验证器 -->
    <validator type="stringlength">
        <param name="fieldName">sno</param>
        <param name="minLength">10</param>
        <param name="maxLength">10</param>
        <param name="trim">true</param>
        <message>学号的长度应该是 10 个字符</message>
    </validator>

    <!-- field 声明 stringlength 验证器 -->
    <field name="myPurchaseCode">
        <field-validator type="stringlength">
            <param name="fieldName">sno</param>
            <param name="minLength">10</param>
            <param name="maxLength">10</param>
            <param name="trim">true</param>
            <message>学号的长度应该是 10 个字符</message>
        </field-validator>
    </field>
</validators>
```

9.3.13 visitor 验证器

有时候,多个 Action 类中可能复合同一个 JavaBean 类。为了避免在每一个 Action 类的验证文件中都编写针对该复合属性的验证,可以把验证信息放到 JavaBean 类的验证文件中,然后在 Action 类的验证文件中直接使用 visitor 验证器去验证该复合属性。表 9-13 给出了 visitor 验证器的参数。

表 9-13 visitor 验证器的参数

参数名称	数据类型	说明
fieldname	String	指示所要验证的 Action 属性名。当使用 validator 元素验证时,需要提供该参数;使用 field 元素验证时,不需要提供该参数
context	String	验证发生的上下文
appendPrefix	Boolean	是否利用 message 子元素为字段错误提示信息添加前缀,默认值为 true

下面介绍一个 visitor 验证器的例子。首先介绍输入页面,如代码 9-14 所示。

代码 9-14 输入页面

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
```

```
        <title>input</title>
    </head>
    <body>
        <s:form action="valid.action">
            <s:radio list="#{1:'type1',2:'type2'}" label="type" name="type"
             value="1"></s:radio>
            <s:textfield name="name" label="name"></s:textfield>
            <s:textfield name="bean.date" label="date"></s:textfield>
            <s:textfield name="bean.number" label="number"></s:textfield>
            <s:textfield name="bean.number2" label="number2"></s:textfield>
            <s:textfield name="bean.text" label="text"></s:textfield>
            <s:submit></s:submit>
        </s:form>
    </body>
</html>
```

代码 9-15 所示为演示 visitor 验证器的 JavaBean 类和 Action 类。

代码 9-15　visitor 验证器示例

```
//ValidatedBean.java
package com.opensymphony.webwork.example;
import java.util.Date;
public class ValidatedBean {
    private String text;
    private Date date=new Date(System.currentTimeMillis());
    private int number;
    private int number2;
    public static final int MAX_TOTAL=12;
    //省略了 get 和 set 方法
}

//ValidatedAction.java
package com.opensymphony.webwork.example;
import com.opensymphony.xwork.ActionSupport;
public class ValidatedAction extends ActionSupport {
    private ValidatedBean bean=new ValidatedBean();
    private String name;
    //省略了 get 和 set 方法

    public String execute() throws Exception{
        return SUCCESS;
    }
}
```

上述代码中定义了一个 Action 类 ValidatedAction.java，该类含有一个普通属性 name 和一个类型为 ValidatedBean 的复合属性 bean。ValidatedBean 的验证文件可以写成代码 9-16 所示的形式。

代码 9-16　ValidatedBean-validation. xml

```
<!--省略了 DOCTYPE -->
<validators>
<field name="text">
<field-validator type="requiredstring">
    <message>Text 不能为空</message>
</field-validator>
</field>
<field name="date">
<field-validator type="date">
    <param name="min">01/01/1970</param>
    <message>日期不合法</message>
</field-validator>
</field>
<field name="number">
<field-validator type="int">
    <param name="min">1</param>
    <param name="max">10</param>
    <message>number 取值范围为${min}到${max}</message>
</field-validator>
</field>
</validators>
```

ValidatedAction 的验证文件可以写成代码 9-17 所示的形式。

代码 9-17　ValidatedAction-validation. xml

```
<!--省略了 DOCTYPE -->
<validators>
<field name="name">
<field-validator type="required">
    <message>name 不能为空!</message>
</field-validator>
</field>

<field name="bean">
<field-validator type="visitor">
    <message>bean: </message>
</field-validator>
</field>
</validators>
```

上述代码利用 visitor 验证器直接调用 ValidatedBean 的验证文件对 ValidatedAction 中的复合属性 bean 进行验证。如果 bean 未通过验证，例如 number 的值输入为 15 时，用户看到的错误提示将是“bean:取值范围为 1 到 10”。错误提示前添加了“bean: ”前缀。

9.3.14　conditionalvisitor 验证器

conditionalvisitor 验证器的实现类是由 visitor 验证器的实现类扩展而来的，除了拥

有 visitor 验证器的所有参数外，还具有表 9-14 给出的参数。

表 9-14　visitor 验证器的参数

参数名称	数据类型	说　　明
expression	Boolean	指定要计算的 OGNL 表达式。该表达式基于 Value Stack 求值。如果计算结果为真，则通过验证，否则验证失败

conditionalvisitor 验证器仅当参数 expression 的值为 true 时，才会调用复合属性的验证文件。例如上节中给出的例子，当希望当用户选中 type1 时，date 属性的取值范围为大于等于 1970 年 1 月 1 日，number 取值范围为[1，10]；当用户选中 type2 时，date 属性的取值范围为大于等于 2000 年 1 月 1 日，number 取值范围为[100，200]。

首先将上节中 ValidatedBean-validation. xml 的文件名修改为 ValidatedBean-u1-validation. xml；然后将 ValidatedBean-u1-validation. xml 复制一份，放到同一个目录中，重新命名为 ValidatedBean-u2-validation. xml，并修改其中的 number 验证范围，使其满足[100，200]；修改 date 验证范围，使其 min 参数为“01/01/2000”。

修改 ValidatedAction-validation. xml，利用 conditionalvisitor 验证器对 ValidatedBean 进行验证，如代码 9-18 所示。

代码 9-18　ValidatedAction-validation. xml（conditionalvisitor 验证器）

```
<validators>
    <field name="name">
        <field-validator type="required">
            <message>name 不能为空!</message>
        </field-validator>
    </field>
    <field name="bean">
        <!-- 当 type=1 时,使用 ValidatedBean-u1-validation.xml 验证 bean 字段 -->
        <field-validator type="conditionalvisitor">
            <param name="expression">type==1</param>
            <param name="context">u1</param>
            <message>U1:</message>
        </field-validator>
        <!-- 当 type=2 时,使用 ValidatedBean-u2-validation.xml 验证 bean 字段 -->
        <field-validator type="conditionalvisitor">
            <param name="expression">type==2</param>
            <param name="context">u2</param>
            <message>U2:</message>
        </field-validator>
    </field>
</validators>
```

上述代码使用 context 参数，在 bean 字段的值不同时分别调用了不同的验证文件去验证复合属性 bean。

9.4 短路验证

如果一个字段被配置了多个字段验证器，当某个验证器验证失败时，如果希望其他验证器不必再验证，可以将 field 元素的 fieldfield-validator 子元素的 short-circuit 属性值设置为 true。如果希望普通验证器在失败时停止其他验证器的工作，可以将 validator 元素的 short-circuit 属性值设置为 true。

下面看一个 Struts 2 官方文档中的例子，如代码 9-19 所示。

代码 9-19 Struts 2 官方文档中短路验证的例子

```
<validators>
    <!-- 字段验证器 1,验证 email 字段 -->
    <field name="email">
        <field-validator type="required" short-circuit="true">
            <message>You must enter a value for email.</message>
        </field-validator>
        <field-validator type="email" short-circuit="true">
            <message>Not a valid e-mail.</message>
        </field-validator>
    </field>
    <!-- 字段验证器 2,验证 email2 字段 -->
    <field name="email2">
        <field-validator type="required">
            <message>You must enter a value for email2.</message>
        </field-validator>
        <field-validator type="email">
            <message>Not a valid e-mail2.</message>
        </field-validator>
    </field>
    <!-- 普通验证器 1 -->
    <validator type="expression">
        <param name="expression">email.equals(email2)</param>
        <message>Email not the same as email2</message>
    </validator>
    <!-- 普通验证器 2 -->
    <validator type="expression" short-circuit="true">
        <param name="expression">email.startsWith('mark')</param>
        <message>Email does not start with mark</message>
    </validator>
</validators>
```

在上述代码中，验证字段 email 的两个字段验证器将 short-circuit 属性设置为 true，验证 email 字段的起始值是否为“mark”的普通验证器 2 的 short-circuit 属性也设置为 true。那么，当其中某个验证器发生短路时，其他验证器会受到什么影响呢？回答这个问题之前，有必要先了解验证器执行的优先顺序。

普通验证器优先于字段验证器。普通验证器按照定义的顺序执行，然后是字段验证器按照定义的顺序执行。

因此，代码 9-19 中的验证器将按照下述顺序执行。

(1) 普通验证器 1

(2) 普通验证器 2

(3) 字段验证器 1

(4) 字段验证器 2

当普通验证器 2 发生短路时，其他验证器不会被执行。当字段验证器 1 中的 required 验证器发生失败时，针对 email 字段的 email 验证器将不会执行，但不会对字段验证器 2 造成任何影响。

9.5　手工验证

某些情况下，当验证条件比较复杂时，使用内置的验证器很难实现验证要求。此时，可以通过手工方式进行验证。ActionSupport 类提供了一个空的 validate()方法，该方法验证 Action 中所有被配置过的方法。当某个数据验证失败时，程序员可以使用 addFieldError()方法添加一条 Field 验证错误信息。Action 实例含有 Field 错误时，Struts 2 会将请求转发到 input 结果视图。如果没有提供 input 结果视图，会打印错误信息到 JSP 页面上，通过<s:fielderror />显示失败的信息。

代码 9-20 演示了通过 Action 类的 validate()方法验证用户输入的例子。

代码 9-20　利用 validate()方法手工验证

```
public class UserManagerAction extends ActionSupport {
    private String userName;
    private String password;
    //省略了 get 和 set 方法
    public String execute() throws Exception {
        //...
        return SUCCESS;
    }
    public String add() throws Exception {
        //...
        return SUCCESS;
    }
    public String del() throws Exception {
        //...
        return SUCCESS;
    }
    public void validate() {
        //验证用户名是否和保留名字相同
        if (userName.equals("admin") || userName.equals("administor")) {
                addFieldError("userName", "用户名不能为保留关键字");
        }
        boolean flag1= false;
        boolean flag2= false;
```

```
            boolean flag3=false;
            for (int i=0; i<password.length(); i++) {
                int ch=password.charAt(i);
                //验证用户名中是否包含小写字母
                if (ch>='a' && ch<='z')
                    flag1=true;
                //验证用户名中是否包含大写字母
                if (ch>='A' && ch<='Z')
                    flag2=true;
                //验证用户名中是否包含数字
                if (ch>='0' && ch<='9')
                    flag3=true;
            }
            //验证密码中是否同时包含大写字母、小写字母和数字
            if (!(flag1 & flag2 & flag3))
                addFieldError("password","密码必须同时包含大写字母、小写字母和数字");
        }
    }
```

当访问 UserAction 类的 execute()、add()和 del()方法时，首先执行 validate()方法。如果验证未通过(含有 field 错误)，Struts 2 直接越过所要调用的方法，返回给用户 Input 结果视图或者原输入页面。

Struts 2 还支持对 Action 指定的方法进行验证。当希望验证 Action 中的方法 xxx()时，可以将验证代码放在 Action 的 validateXxx()方法中。例如，当希望 UserAction 类只在用户访问 add()方法时，Struts 2 才会执行手工验证，可以将代码 9-21 中的 validate()方法的名字改成 validateAdd()，读者可以自己尝试一下。

最后，手工验证和使用 Struts 2 内置的验证器之间并不冲突。使用手工验证时，仍然可以使用内置验证器完成一部分验证工作。

9.6 案例 6：为留言板的注册程序添加输入验证

本节将为留言板的注册程序添加输入验证功能。

9.6.1 自定义字段验证器类

假设实现用户注册的 RegisterAction 类具有如下属性：

```
public class RegisterAction extends ActionSupport {
    private String userName;
    private String password;
    private int gender;
    private String head;
    private String email;
    …
}
```

假设希望保留某些关键字，不允许被注册为用户名。此时，必须编写自己的验证

器类。

1. 编写自定义的字段验证器类

用户自定义的字段验证器可以从 FieldValidatorSupport 类上进行扩展，如代码 9-21 所示。

代码 9-21　保留关键字验证类(ReservedWordValidator.java)

```
package notes.validate;
import com.opensymphony.xwork2.validator.ValidationException;
import com.opensymphony.xwork2.validator.validators.FieldValidatorSupport;

public class ReservedWordValidator extends FieldValidatorSupport {
    private String reservedWord= "";          //保留关键字构成的串,以逗号分隔

    public void setReservedWord(String reservedWord) {
        this.reservedWord= reservedWord;
    }
    //private String[] keyWords= {"manager","note","password"};
    public void validate(Object arg0) throws ValidationException {
        //获取字段名
        String fieldName= this.getFieldName();
        //获取字段值
        String fieldValue= (String) this.getFieldValue(fieldName, arg0);
        //字段值为空,直接返回
        if(null==fieldValue||fieldValue.length()==0)
            return;
        //没有保留关键字,直接返回
        if(reservedWord.length()==0)
            return;
        //获取保留关键字列表
        String[] keyWords= reservedWord.split(",");
        //检查是否含有保留的关键字
        for(String k:keyWords)
            if(fieldValue.contains(k)){
                this.addFieldError(fieldName, arg0);
                return;
            }
    }
}
```

ReservedWordValidator 类中含有一个参数 reservedWord，所有不允许用户使用的关键字都保存在这个参数中，并使用逗号分隔，其值来源于配置验证文件中的 param 元素。

2 注册验证器

验证器必须在注册之后才能使用。内置的验证器已经由 Struts 2 框架注册过了，可以直接使用。如果用户要编写自己的验证器，必须手工注册。

在 src 目录下新建 validators.xml 文件，在该文件中注册 ReservedWordValidator 类，验证器的名字为 reservedWordValidator，如代码 9-22 所示。

代码 9-22 注册自定义的验证器

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC
    "-//Apache Struts//XWork Validator Config 1.0//EN"
    "http://struts.apache.org/dtds/xwork-validator-config-1.0.dtd">
<validators>
    <validator name="reservedWordValidator"
        class="notes.validate.ReservedWordValidator" />
</validators>
```

9.6.2 编写验证文件

在用户输入页面，通常会将性别(gender)和用户头像(head)设置成 radio 按钮组，并提供一个默认值。在验证时，无须对这两个属性进行验证。因此，只需验证 userName、password 和 email 三个属性，如代码 9-23 所示。

代码 9-23 RegisterAction-validation.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC
    "-//Apache Struts//XWork Validator 1.0.3//EN"
    "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">
<validators>
    <!-- 验证用户名 -->
    <field name="userName">
    <field-validator type="required" short-circuit="true">
        <message>用户名不能为空</message>
    </field-validator>
    <!-- 使用 reservedWordValidator 验证用户名是否含有保留关键字 -->
        <field-validator type="reservedWordValidator" short-circuit="true">
        <!-- 保留关键字列表 -->
        <param name="reservedWord">manager,administrator,password</param>
            <message>用户名含有非法关键字</message>
        </field-validator>
        <field-validator type="stringlength">
            <param name="minLength">6</param>
            <message>用户名不能少于 6个字符</message>
        </field-validator>
    </field>
<!-- 验证 email -->
<field name="email">
    <field-validator type="email">
        <message>email 格式不合法</message>
    </field-validator>
</field>
```

```
<!-- 验证密码 -->
<field name="password">
    <field-validator type="stringlength" short-circuit="true">
        <param name="minLength">6</param>
        <message>用户名不能少于 6个字符</message>
    </field-validator>
    <field-validator type="fieldexpression">
        <param name="expression">password==password2</param>
        <message>两次密码不一致</message>
    </field-validator>
</field>
</validators>
```

在利用 fieldexpression 验证 password 时，出现了一个 password2。它对应注册页面的重复密码文本框的 name 属性。由于在 RegisterAction 类中不需要对它处理，因此并没有定义对应的属性名。

同 步 训 练

1. 修改案例 6 中的 RegisterAction 类，利用领域对象 User 接收请求参数，利用 visitor 验证器验证用户输入。

2. 为站内短信系统的所有输入页面添加验证功能。

Chapter 10 第10章 消息处理与国际化

10.1 国际化和本地化

10.1.1 国际化概述

在面向国际市场开发软件应用程序的过程中，程序员必须考虑不同国家文化的差异，让应用程序在无须修改代码的情况下就能支持多种语言和数字格式的显示。例如2012年12月6日，中国人的书写格式为2012-12-06，美国人的书写格式为12/06/2012，英国人的书写格式为06/12/2012。为使应用程序支持不同的字符集和文化标准，如日期、时间、数字和货币格式等，必须使用国际化（Internationalization）技术。国际化，又称I18N，这是因为该单词的第一个字母为I，最后一个字母为N，I和N中间共有18个字母。

将一个国际化程序以本地机器的区域和语言设置显示数字格式和消息化的过程称为本地化（Localization），简称L10N。

Java语言对国际化编程有着很好的支持。由于Struts 2技术建立在Java语言的基础上，因此Struts 2具备对国际化处理的技术。

10.1.2 Java对国际化的支持

一个具备国际化支持的应用程序，必须把针对不同国家和地区的文字消息放到不同的资源文件中。每个资源文件都是一个属性文件，每一个key/value对表示一个资源。资源通常是本地化的String，但也可以是任何Java对象。资源文件以一种层次结构建立，它以一个具有基础名称的一般ResourceBundle开始，然后通过在ResourceBundle的基础名上添加语言和国家或地区标识，构成支持不同国家和地区本地化的资源文件。以下是资源文件命名的3种格式。

```
baseName_language_country.properties
baseName_language.properties
baseName.properties
```

其中，baseName是资源文件的基础名称；language是语言代码，定义在ISO-639中，例如汉语的代码为zh，英语的代码为en，法语的代码为fr等；country是国家代码和地区代码，定义在ISO-3166中，例如中国的代码为CN，美国的代码为US，英国的代码为GB等。

在 Java 程序中，实现国际化需要两个步骤：第一步，选择一个特定的国家、地区和语言环境，实现这一步需要用到 java. util. Locale 类；第二步，根据 Locale，选择用于存储和加载应用程序使用的资源文件，可以通过 java. util. ResourceBundle 类完成。

例如，选择应用于中国的 Locale 对应的代码。

```
Locale locale=new Locale("zh","CN");
```

另外，在 Locale 中还提供了许多 Locale 对象常量，使用这些常量可以简化 Locale 对象的构造。例如，用于表示国家或地区的常量(部分)如下：

```
Locale.CANADA
Locale.CHINA
Locale.FRANCE
Locale.GERMANY
Locale.ITALY
Locale.US
Locale.UK
```

用于表示语言的常量(部分)如下：

```
Locale.CHINESE
Locale.ENGLISH
Locale.FRENCH
Locale.GERMAN
```

在为应用程序选择完国家或地区后，可以根据需要选择使用某一个资源文件。java. util. ResouceBundle 类提供的用于获取资源的方法如下。

(1) public static final ResourceBundle getBundle(String baseName)：根据默认的语言环境选择资源文件。

(2) public static final ResourceBundle getBundle(String baseName，Locale locale)：根据指定的 Locale 获取资源文件。

(3) public final String getString(String key)：读取资源文件中指定关键字的字符串。

(4) public final Object getObject(String key)：读取资源文件中指定关键字的对象。

下面通过一个例子介绍 Java 应用程序实现国际化的方法，程序如代码 10-1 所示。

代码 10-1　TestI18N. java

```
import java.util.Locale;
import java.util.ResourceBundle;

public class TestI18N {
    public static void main(String[] args) {
        //设置本地语言为简体中文
        Locale locale1=Locale.SIMPLIFIED_CHINESE;
        //设置当前语言所用到的资源文件
```

```
            ResourceBundle bundle1=
                ResourceBundle.getBundle("resource",locale1);
            //从资源文件中读取 key=welcome 的消息文本
            System.out.println("bundle1:"+bundle1.getString("welcome"));

            Locale locale2=new Locale("en", "US");
            ResourceBundle bundle2=
                ResourceBundle.getBundle("resource",locale2);
            System.out.println("bundle2:"+bundle2.getString("welcome"));

            Locale.setDefault(Locale.CANADA);      //将默认的 Locale 为 CANADA
            ResourceBundle bundle3=ResourceBundle.getBundle("resource");
            System.out.println("bundle3:"+bundle3.getString("welcome"));
        }
    }
```

TestI18N.java 的功能是从基础名称为 resource 的资源文件中读取字符串 message。locale1 将本地语言设置为简体中文，因此 bundle1 对应的资源文件应该为 resource_zh_CN.properties。locale2 将国家代码设置为美国，因此 bundle2 对应的资源文件应该为 resource_en_US.properties。当将默认国家设置为 Locale.CANADA 后，bundle3 读取的资源文件应该为 resource_en_CA.properties。

下面编写一组资源文件：resource.properties、resource_en_US.properties 和 resource_zh_CN.properties。注意，这里没有提供资源文件 resource_en_CA.properties。

代码 10-2 给出资源文件 resource.properties 的内容。

代码 10-2　resource.properties

```
welcome=Welcome!
```

代码 10-3 给出资源文件 resource_en_US.properties 的内容。

代码 10-3　resource_en_US.properties

```
welcome=Welcome,my friend\!
```

相应的，resource_zh_CN.properties 的内容如代码 10-4 所示。

代码 10-4　resource_zh_CN.properties

```
welcome=\u6B22\u8FCE\u4F60,\u6211\u7684\u670B\u53CB\uFF01
```

实际上，以上信息就是中文的“欢迎你，我的朋友！”。Java 应用中的属性文件在编写时是不能直接写入中文的，就算是写入了中文，读取出来的必然是乱码，因此必须将相应的中文变为 Unicode 编码。

要成功地将一个中文编码变为 Unicode 编码，可以使用 JDK 开发工具包中提供的工具 native2ascii。程序员可以先将资源文件按照 key=value 格式编写一个临时文件，例如

resource_zh_CN. tmp，然后在命令行下执行下述命令。

```
native2ascii resource_zh_CN.tmp resource_zh_CN.properties
```

打开转换获得的 resource_zh_CN. properties，可以看到原文件中所有的非 ASCII 字符都转换成了\uXXXX 形式的 Unicode 编码。

提示：当使用 MyEclipse 开发 Java 应用程序时，可以直接使用 MyEclipse Properties Editor 编辑属性文件，编辑器会将资源中所有非 ASCII 字符自动转换为 Unicode 编码。

编译并运行 TestI18N，得到如下输出：

```
bundle1:欢迎你,我的朋友!
bundle2:Welcome,my friend!
bundle3:Welcome!
```

从运行结果可以看出，由于 bundle3 找不到资源文件 resource_en_CA. properties，因此使用了默认资源文件 resource. properties。所以，假设 Locale 为使用中文的中国内地地区，资源文件的基础名称为 resource，ResourceBundle 的 getBundle()方法将按照如下顺序查找资源文件。

(1) resource_zh_CN. properties

(2) resource_zh. properties

(3) resource. properties

10.1.3 资源的参数化

在上节给出的例子中，资源文件中的资源内容都是固定的。如果希望输出的消息中包含一些动态文本，可以在资源字符串中使用占位符表示出动态文本的位置。占位符的格式为“{编号}”，其中编号的值为 0～9，分别表示第 0 个到第 9 个参数。

在程序中，可以通过 MessageFormat 类对资源进行格式化，为占位符动态设置文本的内容。MessageFormat 是 Format 类的子类，Format 类主要实现格式化操作。除了 MessageFormat 子类外，Format 还有 NumberFormat、DateFormat 两个子类用于对数字显示和日期的显示进行格式化。

例如，有一组资源文件，其中 message-en-US. properties 的内容如代码 10-5 所示。

代码 10-5 message-en-US. properties

```
welcome=welcome {0}, it is {1}.
```

message-zh-CN. properties 的内容如代码 10-6 所示。

代码 10-6 message-zh-CN. properties

```
welcome=\u6B22\u8FCE{0}\uFF0C\u5F53\u524D\u65F6\u95F4\u662F{1}\u3002
```

代码 10-6 中的信息实际上就是“欢迎{0}，当前时间是{1}。”。

代码 10-7 是上述资源文件的测试程序。

代码 10-7 I18NWithParam. java

```
import java.text.MessageFormat;
import java.util.Date;
import java.util.Locale;
import java.util.ResourceBundle;

public class I18NWithParam {
    public static void main(String[] args) {
        Locale locale1=Locale.SIMPLIFIED_CHINESE;
        ResourceBundle bundle1=
            ResourceBundle.getBundle("message", locale1);
        String welcome1=bundle1.getString("welcome");
            //格式化资源,将“李明”传递给第 1个参数,将当前时间传递给第 2个参数
        String msg1=MessageFormat.format(welcome1, "李明",new Date());
        System.out.println(msg1);

        Locale locale2=Locale.US;
        Locale.setDefault(locale2);          //设置默认的 Locale 为美国
        ResourceBundle bundle2=ResourceBundle.getBundle("message");
        String welcome2=bundle2.getString("welcome");
        //格式化资源,将“John”传递给第 1个参数,将当前时间传递给第 2个参数
        String msg2=MessageFormat.format(welcome2, "John",new Date());
        System.out.println(msg2);
    }
}
```

注意：上述代码中的语句

```
Locale.setDefault(locale2);
```

非常重要。当省略该语句时，字符串 msg2 中的日期格式为默认 Locale 下的格式，而不会是美国格式(假设默认 Locale 为中国，默认 Locale 可以通过 Windows 控制面板中的“区域和语言选项”设置)。

编译并运行 I18NWithParam. java，得到如下输出：

```
欢迎李明,当前时间是 12-12-06 下午 3:02。
welcome John, it is 12/06/12 3:02 PM.
```

10.2 Struts 2 对国际化的支持

Struts 2 对 Java 的国际化进行了进一步封装，使得程序员在编写 Action 类和制作 JSP 页面时能够非常便利地实现国际化。

Struts 2 提供了加载国际化资源文件的多种方式，包括全局资源文件、包范围资源文件和 Action 范围资源文件。另外，Struts 2 还允许使用 i18n 标签临时指定资源文件。

1. 全局资源文件

需要被整个 Web 应用程序访问的资源可以放在全局资源文件中。全局资源文件包

括一个默认资源文件和若干个支持不同国家(地区)或语言的资源文件,文件的命名方式与 Java 环境的资源文件命名方式一致。全局资源文件中的资源既可以被 JSP 页面访问,也可以被 Action 类访问。

全局资源文件通常放到 Web 应用程序的 WEB-INF\classes 文件夹下。在使用 MyEclipse 集成环境开发时,可以放到项目的 src 文件夹下。为了让 Struts 2 框架能够找到全局资源文件,还需要在 struts.xml 中配置常量:

```
<constant name="struts.custom.i18n.resources" value="basename" />
```

或者在 src 文件夹中的 struts.properties 文件中加入:

```
struts.custom.i18n.resources=basename
```

其中,basename 为资源文件的基础名称。

2.包范围资源文件

Struts 2 支持对资源文件的模块化管理。当一个 Web 应用涉及的资源比较多时,为了避免全局资源文件过于臃肿,可以将一部分资源放到包范围资源文件中。包范围资源文件应当放到包的根目录中,仅供当前包和子包中的类访问,不能供 JSP 页面和其他包中的类访问。包范围资源文件命名格式为:

(1) package_language_country.properties

(2) package_language.properties

(3) package.properties

其中,package 是固定的写法。

3.Action 范围资源文件

仅供某个 Action 类访问的资源可以放到 Action 范围资源文件中。Action 范围资源文件放到 Action 类所在的目录中,文件命名格式为:

(1) ActionClassName_language_country.properties

(2) ActionClassName _language.properties

(3) ActionClassName.properties

其中,ActionClassName 为 Action 类的简单名称。

4.资源文件查找的顺序

在查找指定 key 的资源时,Struts 2 按照如下顺序搜索资源文件。

(1) 查找与调用的 Action 类同名的资源文件,即对 Action 范围资源文件按照特殊到一般的顺序,查找 ActionClassName_language_country.properties、ActionClassName_language.properties 和 ActionClassName.properties。

(2) 查找所有与 Action 类的基类同名的资源文件,从 ActionSupport.properties 直至 Object.properties。

(3) 查找与所有 Action 类实现的接口同名的资源文件。

(4) 依次查找 Action 类所在包和父包中的资源文件。

(5) 查找全局的资源文件。

10.3 Struts 2 访问国际化资源的方式

10.3.1 在 Action 中访问国际化资源

Struts 2 的 ActionSupport 类中提供了多个重载的 getText()方法,用于访问国际化资源,以下是常用的几个 getText()方法的原型。

(1) public String getText(String aTextName):用于获取 key 为 aTextName 的资源。如果没有找到,将返回 aTextName。

(2) public String getText(String aTextName, String defaultValue):用于获取 key 为 aTextName 的资源。如果没有找到,将返回 defaultValue。

(3) public String getText(String aTextName, List<?> args):用于获取 key 为 aTextName 的资源。参数 args 用于替换资源中的占位符,List 中的第一个元素替换占位符{0},第二个参数替换占位符{1}……

(4) public String getText(String key, String[] args):用于获取 key 为 aTextName 的资源,参数 args 用于替换资源中的占位符,数组中的第一个元素替换占位符{0},第二个参数替换占位符{1}……

另外,ActionSupport 类还提供了一个 getTexts()方法。

```
public ResourceBundle getTexts(String aBundleName)
```

用于获得名为 aBundleName 的 ResourceBundle 对象。通过该对象,可以访问基础名称为 aBundleName 的资源文件中的资源。

下面通过一个例子讨论如何在 Action 中访问国际化资源。

修改前面给出的 resource_zh_CN. properties 资源文件,如代码 10-8 所示。

代码 10-8 修改后的 resource_zh_CN. properties

```
welcome=\u6B22\u8FCE\u4F60,\u6211\u7684\u670B\u53CB\uFF01
hello=\u6B22\u8FCE{0}\uFF0C\u73B0\u5728\u65F6\u95F4{1}\u3002
userName=\u7528\u6237\u540D
password=\u5BC6\u7801
submit=\u63D0\u4EA4
```

上述内容实际如下:

```
welcome=欢迎你,我的朋友!
hello=欢迎{0},现在时间{1}。
username=用户名
password=密码
submit=提交
```

修改前面给出的 message_zh_CN. properties 资源文件,增加如下内容。

```
hello=\u4F60\u597D\u5417\uFF1F
```

上述内容实际如下：

```
hello=你好吗？
```

在 struts.xml 中将 resource.properties 配置为全局资源文件。代码 10-9 定义了一个 Action 类，并演示了在 Action 类中访问资源文件的方法。

代码 10-9　I18NTestAction.java

```
package org.ch11.action;
import java.util.Date;
import java.util.ResourceBundle;
import com.opensymphony.xwork2.ActionSupport;
public class I18NTestAction extends ActionSupport {
    public String execute(){
        String welcome=getText("welcome");                    //①
        System.out.println("welcome="+welcome);
        String hello=getText("hello",
            new String[]{"李明",new Date().toString()});       //②
        System.out.println("hello="+hello);

        ResourceBundle bundle=getTexts("message");            //③
        String msgHello=bundle.getString("hello");            //④
        System.out.println("message::hello="+msgHello);
        return SUCCESS;
    }
}
```

语句①获取了全局资源文件中 Key 为 welcome 的资源。

语句②在获取全局资源文件中 Key 为 hello 的资源时，利用字符串数组向资源传递了两个参数，用于替换资源中的占位符。

语句③利用 getTexts()方法读取基础名称为 message 的资源文件，并利用④从中获取 Key 为 hello 的资源。

访问 I18NTestAction，获得如下输出：

```
message=欢迎你,我的朋友!
hello=欢迎李明,现在时间 Fri Dec 07 09:50:13 CST 2012。
message::hello=你好吗？
```

10.3.2　在 JSP 页面中访问国际化资源

在 JSP 页面的一般文本中显示国际化资源，需要使用 Struts 2 提供的 text 标签和 i18n 标签。

1. text 标签

text 标签用于显示国际化资源，相当于调用了 getText()方法的 property 标签。text 标签常用的属性如表 10-1 所示。

表 10-1 text 标签的属性

属 性 名	是否必需	类 型	说 明
name	是	String	所要访问的国际化资源的 Key
searchValueStack	否	Boolean	如果没有找到指定消息，将从 ValueStack 中搜索，此时相当于一个 property 标签。默认值为 true
var	否	String	允许用户根据该属性引用 var

例如，下面的 text 标签将访问 Key 为 welcome 的资源。

```
<s:text name="welcome"></s:text>
```

如果希望访问带有占位符的资源，可以利用 param 标签为 text 标签传递参数。例如，访问上节中的 hello 标签，可以使用下面的形式。

```
<s:text name="hello">
    <s:param>李明</s:param>
    <s:param>2012-12-07 19:20:00</s:param>
</s:text>
```

另外，还可以使用动态值作为 text 标签的属性，例如：

```
<s:text name="hello">
    <s:param><s:property value="userName"/></s:param>
    <s:param><s:property value="now"/></s:param>
</s:text>
```

在利用 text 标签显示国际化资源时，所读取的资源文件取决于 Action 类的 Bundle。

2. i18n 标签

i18n 标签用于读取一个 bundle，并将其放到 ValueStack 上。通过 i18n 标签，程序员可以让 text 标签显示任意一个 bundle 对应的资源文件中的资源。i18n 标签仅有一个 name 属性，该属性用于指定所要读取的资源文件的基础名称。

下面在 10.3.1 小节的基础上，为 I18NTestAction.java 编写一个结果页面，如代码 10-10 所示。

代码 10-10 在 JSP 页面中输出国际化资源

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<html>
    <head>
        <title>Internationalization</title>
    </head>
    <body>
        <!--访问全局资源文件中的 welcome 资源 -->
        <s:text name="welcome"></s:text>
        <br/>
        <!--访问全局资源文件中的 hello资源,并传递两个参数 -->
        <s:text name="hello">
```

```
            <s:param>李明</s:param>
            <s:param>2012-12-07 19:20:00</s:param>
        </s:text>
        <br/>
        <!-- i18n标签访问 message资源文件 -->
        <s:i18n name="message">
            <!--访问基础文件名为 message的资源文件中的 hello资源 -->
            <s:text name="hello"></s:text>
        </s:i18n>
    </body>
</html>
```

代码 10-10 中给出了详细的注释，在此不再赘述。在 struts.xml 配置 I18NTestAction.java 和该页面后，在浏览器中访问 Action，页面显示内容如下：

```
欢迎你,我的朋友!
欢迎李明,现在时间 2012-12-07 19:20:00。
你好吗?
```

10.3.3　在表单标签的属性中访问国际化资源

除了可以在 JSP 页面中输出一般的国际化资源外，也可以对表单中的标签进行国际化。在表 6-15 中给出的表单标签的通用属性中有一个属性 key，当为表单标签指定该属性时，Struts 2 将使用国际化资源文件中对应的资源作为 HTML 表单元素的 name、label 和 value。

代码 10-11 给出了一个在表单标签中访问国际化消息的例子。

代码 10-11　i18nForm.jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<html>
    <head>
        <title>I18N Form</title>
    </head>
    <body>
        <s:form action="i18nForm">
            <s:textfield key="userName"></s:textfield>
            <s:password key="password"></s:password>
            <s:submit key="submit"></s:submit>
        </s:form>
    </body>
</html>
```

由于上述代码中没有为 textfield 标签和 password 标签指定属性 name，Struts 2 会直接用属性 Key 作为属性 name。在浏览器中查看 i18nForm.jsp，效果如图 10-1 所示。

最后，提醒大家，只有当 JSP 的表单和表单标签的主题不为 simple 时，才能使用表单

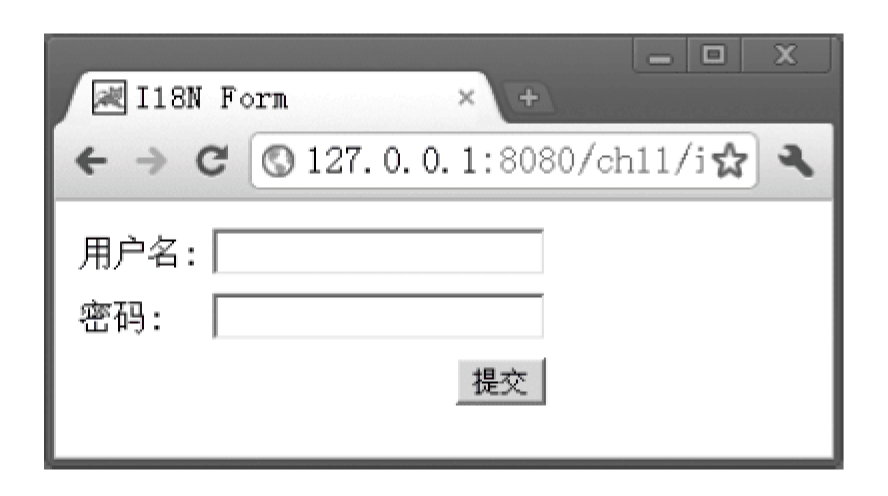


图 10-1 表单标签访问国际化资源

标签的 key 属性访问国际化资源。

10.4 案例 7：为留言板程序添加国际化支持

本节将为留言板程序添加中、英文两种语言支持，允许用户通过一个下拉列表框自由选择需要使用的语言环境。一旦用户选择了某种语言，在接下来的每个页面中都将使用该种语言浏览。

为实现上述功能，需要借助 Struts 2 提供的 i18n 拦截器。在执行 Action 的方法之前，i18n 拦截器的 intercept()方法先查找一个名为 requested_locale 的参数。如果能够找到该参数，i18n 将其转换成一个 Locale 对象，并保存在 session 的 WW_TRANS_I18N_LOCALE 参数中。可以通过修改该参数中保存的 Locale 对象，改变应用程序运行的语言环境。

10.4.1 编写资源文件

限于篇幅，本节只给出留言板注册页面的国际化支持。

代码 10-12 所示是注册页面的简体中文的资源文件。注意，为了方便读者阅读，该文件没有转换成 Unicode 编码形式，读者可自行使用 native2ascii 工具进行转换。

代码 10-12 resource_zh_CN.txt

```
title=留言板——注册
userName=用户名
password=密码
repassword=重复密码
email=邮箱
gender=性别
gender.male=男
gender.female=女
photo=选择头像
submit=注册
reset=重填

chinese=中文版
english=英文版
selectlanguage=请选择语言
```

代码 10-13 所示是注册信息的英文资源文件。

代码 10-13　resource_en_US. properties

```
title=Notes--Welcome to register
userName=UserName
password=Password
repassword=RePassword
email=Email
gender=Gender
gender.male=Male
gender.female=Female
photo=Choose your avatar
register=Register
submit=Register
reset=Reset

chinese=Chinese
english=English
selectlanguage=Please select language
```

10.4.2　JSP 页面的国际化

为了便于用户选择语言环境，可以在注册页面添加一个下拉列表框，并在其中列出留言板程序所支持的语言环境。为此，采用一个 Map 填充下拉框。其中，Map 的 key 为 java. util. Locale 常量，value 为国际化资源。

```
<s:select name="request_locale"
list="#{@java.util.Locale@CHINESE:getText('chinese'),
    @java.util.Locale@US:getText('english')}"
    key="selectlanguage"
    value="#session_locale=null ? locale : #session_locale"/>
```

这里使用表示中国的 java. util. Locale@CHINA 常量(字符串的值为 zh_CN)对应前面定义的资源文件 resource_zh_CN. properties，表示美国的 java. util. Locale@US 常量(字符串的值为 en_US)对应前面定义的资源文件 resource_en_US. properties。

另外，为了获取应用程序使用的语言环境，可以读取 session 对象的 WW_TRANS_I18N_LOCALE 参数，并将该参数的值作为下拉列表框当前选中的值。

当用户选择某个语言环境后，下拉列表框的值将作为 request_locale 参数传递给 Struts 2 框架，由 i18n 拦截器根据该参数设置应用程序的语言环境。

代码 10-14 给出了注册页面完整的代码。

代码 10-14　注册页面(reg. jsp)

```
<%@page language="java" pageEncoding="UTF-8"%>
<%@taglib uri="/struts-tags" prefix="s"%>
<html>
<head>
<title><s:text name="title"/></title>
<script type="text/javascript">
function lang_onChanged() {
```

```
        document.getElementById("langForm").submit();
    }
    </script>
    </head>
    <body>
    <!--将 Session对象的 WW_TRANS_I18N_LOCALE参数保存在 session_locale中 -->
    <s:set name="session_locale" value="#session['WW_TRANS_I18N_LOCALE']"/>
    <!--下面的 form用于将下拉列表框选项的改变提交给服务器 -->
    <form action="<s:url/>" id="langForm" style="background-color:#9e9e9e;">
        <s:select name="request_locale"
        list="#{@java.util.Locale@CHINESE:getText('chinese'),
            @java.util.Locale@US:getText('english')}"
                key="selectlanguage"
                value="#session_locale=null ? locale : #session_locale"
                onchange="lang_onChanged()" />
    </form>
    <!--      用户注册表单      -->
        <s:form name="regForm" action="register.action" method="post">
            <s:textfield key="userName"></s:textfield>
            <s:password key="repassword"></s:password>
            <s:textfield key="email"></s:textfield>
            <s:radio list="#{1:getText('gender.male'),
                0:getText('gender.female')}" key="gender" value="1"></s:radio>
            <s:radio list="#{'1.gif':'<img src=images/head/1.gif >',
                '2.gif':'<img src=images/head/2.gif> ',
                '3.gif':'<img src=images/head/3.gif> ',
                '4.gif':'<img src=images/head/4.gif> ',
                '5.gif':'<img src=images/head/5.gif><br/> ',
                '6.gif':'<img src=images/head/6.gif> ',
                '7.gif':'<img src=images/head/7.gif> ',
                '8.gif':'<img src=images/head/8.gif> ',
                '9.gif':'<img src=images/head/9.gif> ',
                '10.gif':'<img src=images/head/10.gif> '
            }" name="head" value="'1.gif'" key="photo"></s:radio>
            <s:submit key="submit"></s:submit>
        </s:form>
    </body>
    </html>
```

在上述代码中，当列表框的选项发生改变时，利用 JavaScript 自动将 langForm 表单提交给服务器。

图 10-2 所示为注册页面在语言环境为中文时的显示效果。当选择以英文浏览时，注册页面的显示效果如图 10-3 所示。

注意：由于页面需要访问 session 对象，因此不能在浏览器中直接访问 JSP 页面，而应该通过 Action 浏览。

细心的读者可能注意到了，在图 10-3 给出的英文版注册页面中出现了一条不协调的文字——“用户名不能为空”，这是在前面的章节为 userName 字段配置的验证错误信息。显然，在英文版的页面中出现这样的文字是不合适的，下面将研究如何国际化校验信息。

图 10-2　注册页面(中文环境)

图 10-3　注册页面(英文环境)

10.4.3　校验信息的国际化

第 8 章介绍了如何利用 Struts 2 的校验框架来验证用户输入的信息,当用户输入信息不符合应用的约束时,validation 拦截器会把验证文件中读取到的提示放入 Field 级别或 Action 级别的错误消息中,然后在 JSP 页面利用 fielderror 标签或 actionerror 标签提示给用户。在第 8 章的例子中,所有的错误提示都硬编码在验证文件中。对于一个支持国际化的应用程序来说,显然这是不符合要求的。下面通过留言板注册程序来研究如何实现验证信息的国际化。

首先,编写提供验证信息的国际化资源文件。由于这部分资源仅在注册时使用,所以应该使用 Action 范围的资源文件进行存储。代码 10-15 给出了注册验证信息的中文

部分，为了便于用户阅读，没有将其转换成 Unicode 编码。

代码 10-15 RegisterAction-zh-CN. properties 的内容

```
userName.empty=用户名不能为空
userName.invalid=用户名含有非法关键字
userName.minLength=用户名不能少于 6个字符
email.invalid=email 格式不合法
password.minLength=密码不能少于 6个字符
password.different=两次密码不一致
```

RegisterAction-en-US. properties 是验证信息的英文资源文件，如代码 10-16 所示。

代码 10-16 RegisterAction-en-US. properties

```
userName.empty=username cannot be empty!
userName.invalid=username is invalid!
userName.minLength=The username length can not be less than 6 characters!
email.invalid=email is invalid!
password.minLength=The password length can not be less than 6 characters!
password.different=Password does not match the confirm password!
```

接下来，利用上述两个资源文件实现验证信息的国际化。

Struts 2 验证文件中的 message 标签用于指定产生错误时的提示信息，该标签有一个 key 属性。当将 key 属性的值指定为国际化资源中的某个资源的 key 时，就能够实现验证信息的国际化。

代码 10-17 给出了添加了国际化支持的留言板注册程序的验证文件。

代码 10-17 RegisterAction-validation. xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC
    "-//Apache Struts//XWork Validator 1.0.3//EN"
    "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">
<validators>
    <!-- 验证用户名 -->
    <field name="userName">
        <field-validator type="required" short-circuit="true">
            <message key="userName.empty"></message>
        </field-validator>
        <!-- 使用 reservedWordValidator 验证用户名是否含有保留关键字 -->
        <field-validator type="reservedWordValidator"
                short-circuit="true">
            <!-- 保留关键字列表 -->
            <param name="reservedWord">manager,administrator,password
            </param>
            <message key="userName.invalid"></message>
        </field-validator>
        <field-validator type="stringlength">
            <param name="minLength">6</param>
            <message key="userName.minLength"></message>
```

```
        </field-validator>
    </field>

    <!-- 验证 email -->
    <field name="email">
        <field-validator type="email">
            <message key="email.invalid"></message>
        </field-validator>
    </field>

    <!-- 验证密码 -->
    <field name="password">
        <field-validator type="stringlength" short-circuit="true">
            <param name="minLength">6</param>
            <message key="password.minLength"></message>
        </field-validator>
        <field-validator type="fieldexpression">
            <param name="expression">password==password2</param>
        <message key="password.different"></message>
        </field-validator>
    </field>
</validators>
```

注意：当同时为 message 标签指定值和 key 属性时，起作用的为 Key 属性。例如下面的代码：

```
<message key="userName.invalid">用户名中包含非法的关键字!</message>
```

当发生错误时，Struts 2 提示给用户的是资源 userName.invalid 的内容，而不会将字符串“用户名中包含非法的关键字!”提示给用户。

图 10-4 给出了最终的注册页面的英文版。从图中可以看到，验证信息已经实现了国际化。

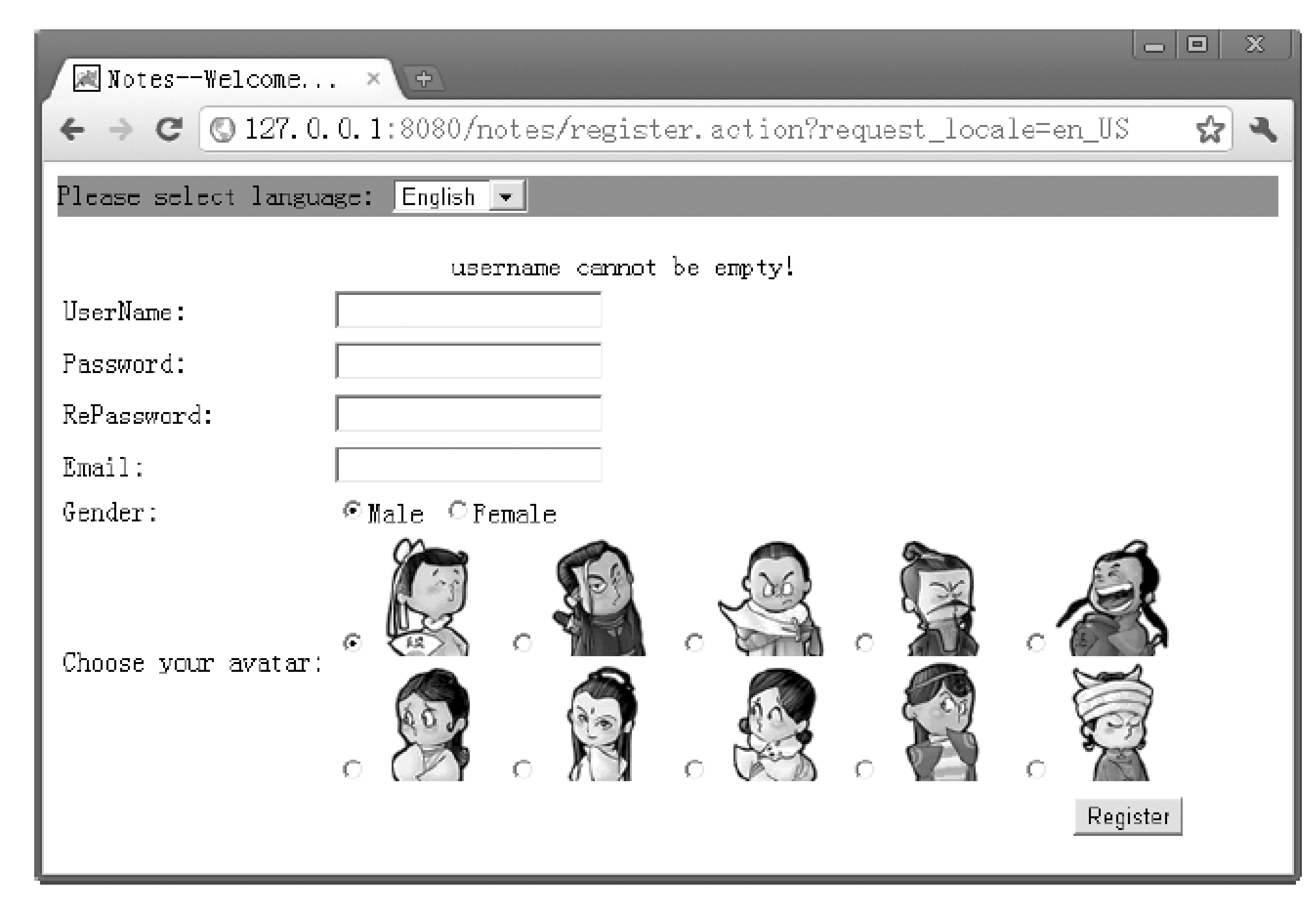


图 10-4 留言板注册页面的英文版

同 步 训 练

1. 编程测试资源文件的加载顺序。

2. 为留言板程序的登录页面、留言页面和查看留言内容页面提供国际化支持。

3. 为站内短信系统添加国际化支持。要求对国际化资源实现模块化处理，并根据资源使用范围的不同，将其放入不同范围的资源文件。

Chapter 11 第11章 类型转换

11.1 类型转换概述

在 Web 应用中，页面提交的数据是以请求参数的形式发送给服务程序的，每一个请求参数都是字符串类型的数据。在服务程序端，必须将这些数据转换成特定的数据类型，例如 java.model.Date 或 float 类型，然后进行处理。

在传统的 JSP 编程中，类型转换的过程通常需要程序员编码完成。而在 Struts 2 中，基本类型和字符串之间的类型转换能够通过内置的类型转换器自动完成。只要用户输入的数据符合 Action 中相应属性的类型要求，在 execute()方法执行之前，Struts 2 能够将其自动转换为合适的数据类型。例如，对于留言板程序中的用户注册页面，在输入了用户名、密码和性别，并点击提交之后，Struts 2 自动将提交的性别由字符串类型转换成为 Int 类型，然后将转换后的 Int 数据赋值给 Action 的 User.gender 属性(注意，User.gender 的类型为 Int 型，其中“0”表示男，“1”表示女)。

11.1.1 Struts 2 内置的类型转换器

Struts 2 内置的类型转换器提供了对类型转换的无缝支持，能够处理绝大多数的需求，例如将 String 转换为 Int 等简单类型，以及将 String 转换到 Date 等对象类型。Struts 2 内置的类型转换器实现 String 类型和下述类型的自动转换。

(1) int/Integer，float/Float，long/Long，double/Double：在字符串和数值型数据之间转换。

(2) boolean/Boolean：在字符串和布尔型数据之间转换。

(3) char/Character：在字符串和字符之间转换。

(4) date：在字符串和日期数据之间转换。转换时，将根据 Locale 设置，利用 SHORT 格式进行输入和输出处理。

(5) BigInteger/BigDecimal：在字符串和大整型数据/大浮点数型数据之间转换。

(6) Enumeration：在字符串和枚举类型数据之间转换。

(7) Array：在字符串和数组数据之间转换。在转换时，Struts 2 将尝试使用数组元素类型对应的转换器对每个元素进行转换。

(8) Collection：在字符串和集合类型数据之间转换。在转换时，如果无法确定元素的类型，Struts 2 将该元素视为 String 类型，并且创建一个 ArrayList 来存放元素。

下面举一个 Struts 2 对枚举类型数据自动转换的例子。

代码 11-1 给出了一个枚举类 SchoolEnum 的定义。

代码 11-1　枚举类 SchoolEnum

```
public enum SchoolEnum {
    计算机学院,信控学院,理学院,石化学院;
}
```

代码 11-2 给出了使用 SchoolEnum 的 Action 类。

代码 11-2　SchoolAction

```
import org.ch11.model.SchoolEnum;
import com.opensymphony.xwork2.ActionSupport;
public class SchoolAction extends ActionSupport {
    SchoolEnum school;
    //省略了 school 的 getter 和 setter 方法
    public String execute() throws Exception {
        return SUCCESS;
    }
}
```

代码 11-3 给出了一个 school.jsp 页面。其中,列表框 school 绑定到 SchoolAction 类的 school 属性上。

代码 11-3　school.jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head><title>选择学院</title></head>
    <body>
        <s:form action="school">
            <s:select list="{'计算机学院','信控学院','理学院','石化学院'}"
            label="所在学院" name="school"></s:select>
            <s:submit></s:submit>
        </s:form>
        你所在的学院为:<s:property value="school"/>
    </body>
</html>
```

当用户在 school.jsp 页面选择所在学院并提交后,Struts 2 框架将请求数据填充给 SchoolAction 的 school 属性时,会调用 xWork 提供的 DefaultTypeConverter 类将数据转换成 SchoolEnum 类型。同样,在返回结果页面时,Struts 2 将 SchoolEnum 类型的 school 转换成 String 类型数据进行响应。

在此省略了上述例子的 struts.xml 配置,读者可以自行补充完成。

11.1.2　类型转换时装配对象的原则

当利用表单为一个 JavaBean、List 和 Map 对象输入数据时，Struts 2 的类型转换实现了将客户输入直接装配成一个对象，而不需要使用基本类型或字符串类型的表单参数值作为中间值，然后由程序员在 Action 的 execute()方法中把这些中间值组装成完整的对象。Struts 2 在类型转换时装配对象的原则如下。

（1）当表单输入的是一个组合的 OGNL 表达式时，Struts 2 首先计算出该表达式的值，然后利用该值自动创建和装配实际对象。

（2）当表单标签的 name 对应 Action 类中 JavaBean 类型的属性时，Struts 2 自动创建一个 JavaBean 对象，并为其赋值。需要注意的是，Struts 2 只能自动创建和装配遵守 JavaBean 规范的对象，也就是说，该对象必须提供无参构造函数，以及为每个 public 类型的属性提供 getter 和 setter 方法。例如，某个 Action 类有一个名为 note 的 JavaBean，如果希望 Struts 2 创建 note 对象，那么 Action 类必须提供一个 setPerson()方法。另外，当为 note. title 赋值时，调用 getNote(). setTitle()方法。

（3）当表单中具有多个同名的标签时，Struts 2 尝试将其装配为一个数组或 List 对象。

在 Java 应用开发过程中，程序员经常会遇到 NullPointerException 的问题。产生这个问题的原因是引用了一个没有初始化的对象。Struts 2 在进行类型转换过程中，如果发现没有初始化的对象，会自动建立一个空的 Object 的引用。这一功能的支持是由 Parameter 拦截器完成的。该拦截器在开始处理参数之前，将 Action Context 中一个名为 CREATE_NULL_OBJECTS 的键值设置为 true。这样，出现 NullPointerException 异常的 OGNL 表达式将被自动临时中断，系统将通过创建所需对象的方法来自动尝试解决空值引用。

Struts 2 在处理空值引用时，有如下规则。

（1）当属性被声明为 Collection 或者 List 时，返回一个 ArrayList 对象来赋给空引用。

（2）当属性声明为 Map 时，返回一个 HashMap 对象赋给空引用。

（3）当属性声明为一个简单带有无参数构造方法的 JavaBean 时，调用无参数的构造方法生成这个 JavaBean 的实例并赋给空引用。

11.2　复杂对象类型的转换

Struts 2 对于数组、List 和 Map 等集合类型的转换提供了很好的支持。为了支持集合类型的转换，需要使用泛型技术。本节对常用集合类型的转换进行详细讨论。

11.2.1　数组和 List 的类型转换

对于需要快速录入批量数据的场合，可以使用数组或 List 接收用户输入的数据。下面讨论一个利用 List 保存部门信息的例子。代码 11-4 给出了一个 JavaBean 类 Department。

代码 11-4 Department.java

```
package org.ch11.model;
public class Department {
    private int deptno;
    private String dName;
    private String loc;
    //省略了 deptno、dName 和 loc 的 getter 和 setter 方法
}
```

提示：在使用 myEclipse 自动生成 Department 类的各个属性的 getter 和 setter 方法时，dName 的 getter 和 setter 方法被命名为 getdName()和 setdName()。使用时，Java 运行环境无法访问 dName 属性，必须将其修正为 getDName()和 setDName()。因此，尽可能不要以首字母为小写字母，第 2 个字母为大写字母的形式命名 Java 属性。如果非要这么做，一定要手工修改 myEclipse 自动生成的 getter 和 setter 方法名。

代码 11-5 给出的 DepartmentAction 类利用 List 对象保存了添加的部门信息。DepartmentAction 类中包含两个 List 属性，均使用了泛型技术。其中，memo 中存放的是简单数据类型的对象，depts 中存放的是 JavaBean。注意，depts 和 memo 并没有实例化，当 Struts 2 为其填充数据时，将自动进行实例化。

代码 11-5 DepartmentAction.java

```
package org.ch11.action;
import java.model.List;
import org.ch11.model.Department;
import com.opensymphony.xwork2.ActionSupport;
public class DepartmentAction extends ActionSupport {
    private List<Department> depts;
    private List<String> memo;
    //省略了 depts 和 memo 的 getter 和 setter 方法
    public String execute() {
        return SUCCESS;
    }
}
```

代码 11-6 是用于添加部门信息的 JSP 页面，页面提交的结果将交由 DepartmentAction 处理。请注意页面中为 DepartmentAction 的 depts 和 memo 属性提供数据的 TextField 的 name 的写法。

代码 11-6 添加部门的 JSP 页面(addDept.jsp)

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
```

```
        <title>添加部门</title>
    </head>
    <body>
        <s:form action="deptAdd" theme="simple" method="post" namespace="/list">
            <table>
            <tr>
                <th>部门编码</th>
                <th>部门名称</th>
                <th>办公地点</th>
                <th>备　　注</th>
            </tr>
            <s:iterator value="new int[2]" status="st">
                <tr>
                    <td>
                        <s:textfield name="%{'depts['+#st.index+'].deptno'}"/>
                    </td>
                    <td>
                        <s:textfield name="%{'depts['+#st.index+'].dName'}"/>
                    </td>
                    <td>
                        <s:textfield name="%{'depts['+#st.index+'].loc'}"/>
                    </td>
                    <td>
                        <s:textfield name="memo"></s:textfield>
                    </td>
                </tr>
          </s:iterator>
            <tr>
                <td colspan="3">
                <s:submit></s:submit>
                </td>
            </tr>
            </table>
        </s:form>
    </body>
</html>
```

注意：在为 Action 类的存放简单类型数据的 List 属性输入值时，textfield 的 name 属性直接写成对应的 Action 属性名即可；如果为 Action 类的存放 JavaBean 类型的属性输入值时，textfield 的 name 属性一定要写成。

```
BeanName[index].attributeName
```

例如，代码 11-6 中的 depts[0].dName、depts.[1].dName 等。

图 11-1 所示为 addDept.jsp 在浏览器中的输出效果。

代码 11-7 为对应的结果页面的代码。

图 11-1 addDept. jsp

代码 11-7 deptList. jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
        <title>您输入的部门是</title>
    </head>
    <body>
        <table>
            <tr>
                <th>部门编码</th>
                <th>部门名称</th>
                <th>办公地点</th>
                <th>备    注</th>
            </tr>
            <s:iterator value="depts" status="st">
                <tr>
                    <td>
                        <s:property value="deptno"></s:property>
                    </td>
                    <td>
                        <s:property value="dName"></s:property>
                    </td>
                    <td>
                        <s:property value="loc"></s:property>
                    </td>
                    <td>
                        <s:property value="memo[#st.index]"/>
                    </td>
                </tr>
            </s:iterator>
        </table>
    </body>
</html>
```

在上述代码中，利用 Struts 2 的 iterator 标签对 DepartmentAction 的 depts 和 memo 进行迭代输出。注意，由于 memo 和 depts 为两个不同的 List 对象，当使用同一个 iterator 迭代时，一个 List 直接使用 iterator 的 value 属性控制，而另一个 List 只能通过下标控制。例如，上述代码中的

```
<s:property value="memo[#st.index]"/>
```

利用 IteratorStatus 对象的 index 属性直接存取。

使用数组和 List 极其相似，在提交页面上的写法相同；在 Action 类中除了声明外，也没有太大的区别，在此不再赘述。读者可自行将上述例子修改成数组形式。

图 11-2 所示为 addDept.jsp 提交后的结果页面。



图 11-2　addDept.jsp 提交后的结果页面

11.2.2　Map 的类型转换

Map 以 key/value 的形式存储数据，和数组非常相似。不同的是，数组的下标为整型数据，而 Map 的下标（Map 的 Key）是一个对象。

有时候，Web 应用程序可能需要使用 Map 对象来接收用户输入的数据。代码 11-8 是在代码 11-5 的基础上修改而来的，其中的 depts 属性和 memo 属性的类型被修改成了 Map。

代码 11-8　DepartmentAction.java（使用 Map）

```
package org.ch11.map.action;
import java.util.Map;
import org.ch11.model.Department;
import com.opensymphony.xwork2.ActionSupport;
public class DepartmentAction extends ActionSupport {
    private Map<String, Department> depts;
    private Map<String, String> memo;
    //省略了 depts 和 memo 的 getter 和 setter 方法

    public String execute() {
        return SUCCESS;
    }
}
```

与使用数组和 List 类似，Struts 2 在为 Map 对象填充数据时，如果发现 Map 对象为空，也会自动实例化。

代码 11-9 是在代码 11-6 的基础上修改而来的。

代码 11-9 addDept.jsp(使用 Map)

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
        <title>添加部门</title>
    </head>
    <body>
        <s:form action="addDept" theme="simple" method="post" namespace="/map">
        <table>
        <tr>
            <th>部门编码</th>
            <th>部门名称</th>
            <th>办公地点</th>
            <th>备    注</th>
        </tr>
        <s:iterator value="new int[2]" status="st">
            <tr>
                <td>
                    <s:textfield name="%{'depts['+#st.index+'].deptno'}"/>
                </td>
                <td>
                    <s:textfield name="%{'depts['+#st.index+'].dName'}"/>
                </td>
                <td>
                    <s:textfield name="%{'depts['+#st.index+'].loc'}"/>
                </td>
                <td>
                    <s:textfield name="%{'memo['+#st.index+']'}"/>
                </td>
            </tr>
          </s:iterator>
            <tr>
                <td colspan="3">
                    <s:submit></s:submit>
                </td>
            </tr>
        </table>
        </s:form>
    </body>
</html>
```

本例中直接使用 IteratorStatus 对象的 index 属性值作为 Map 对象的 Key。对照代码代码 11-6,可以看出存储 JavaBean 的 List 对象和 Map 对象,在数据输入页面中,对应

的 textfield 的 name 属性的写法是一样的;但是存储简单数据的 List 对象和 Map 对象的写法是有区别的,为 Map 对象输入值时,一定要将 textfield 标签的 name 属性写成 ActionName[key]的形式。

代码 11-10 所示为 addDept.jsp 对应的结果输出页面。

代码 11-10　ListDept.jsp(修改成 Map)

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
    <head>
        <title>您输入的信息</title>
    </head>
    <body>
        <table>
            <tr>
                <th>Key</th>
                <th>部门编码</th>
                <th>部门名称</th>
                <th>办公地点</th>
                <th>备　　注</th>
            </tr>
            <s:iterator value="depts">
                <tr>
                    <td>
                        <s:property value="key"></s:property>
                    </td>
                    <td>
                        <s:property value="value.deptno"></s:property>
                    </td>
                    <td>
                        <s:property value="value.dName"></s:property>
                    </td>
                    <td>
                        <s:property value="value.loc"></s:property>
                    </td>
                    <td>
                        <s:property value="memo[key]"/>
                    </td>
                </tr>
            </s:iterator>
        </table>
    </body>
</html>
```

在本例中,在利用 iterator 控制输出 Map 对象 depts 的同时,也控制了另一个 Map 对象 memo 的输出,注意写法的不同。读者也可以将该例与代码 11-7 进行比较,看看有

何区别。

图 11-3 所示为 deptList.jsp 在浏览器中的输出。其中，addDept.jsp 中输入的各数据与图 11-1 中是一样的。请读者将图 11-2 与图 11-3 进行比较，看看输出的次序有何不同。



图 11-3 deptList.jsp 在浏览器中的输出

11.3 自定义类型转换器

11.3.1 开发自定义类型转换器

有时候，Struts 2 自带的类型转换器可能无法满足应用的需求，例如它无法将货币￥12,365,487.00 转换成 java.lang.Number 或其他数值型类型，也无法将空间坐标(x,y,z)转换成某种合适的 Java 数据类型。这时，程序员必须开发自己的类型转换器。

自定义类型转换器，需要实现 ognl.TypeConvert 接口，也可以扩展该接口的实现类 DefaultTypeConverter 或者 StrutsTypeConverter。

ognl.TypeConvert 接口只有一个方法，原型如下：

```
public Object convertValue(Map<String,Object> context,Object target,
            Member member,String propertyName,Object value,Class toType)
```

其中，各参数说明如下。

(1) context：将在其中进行类型转换的上下文环境。通过该参数，程序员可以访问 Value Stack 中的资源。

(2) target：将在其中对属性设置的目标对象。

(3) member：将被设置的类成员、构造函数或属性。

(4) propertyName：将被设置的属性名。

(5) value：将被转换的值。

(6) toType：将被转换成的目标类型。

DefaultTypeConverter 类实现了 TypeConvert 接口，并提供了 convertValue()方法的简化版本。

```
public Object convertValue(Map<String,Object> context,Object value,
                    Class toType)
```

代码 11-11 实现了将形如"￥12,365,487.00"的字符串与 Double 类型之间的转换。

代码 11-11　CurrencyConverter.java

```
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.model.Map;
import ognl.DefaultTypeConverter;
public class CurrencyConverter extends DefaultTypeConverter {
    public Object convertValue(Map context, Object value, Class toType) {
        if(toType==double.class){              //由 String 类型转换成 double 类型
            String[] s= (String[])value;
            String v=s[0].substring(1);
            v=v.replace(",", "");
            return Double.parseDouble(v);
        }
        else if(toType==String.class){         //由 double 类型转换成 String 类型
            NumberFormat fmt=new DecimalFormat("￥#,##0.00");
            return fmt.format(value);
        }
        return null;
    }
}
```

利用 CurrencyConverter 转换器，可以将客户端提交的字符串"￥12,365,487.00"转换成 Double 类型数据 12365487.00；并且当向客户端输出 Double 型数据时，CurrenyConverter 转换器将再次启用，将 12365487.00 转换成"￥12,365,487.00"。

通过代码 11-11 可以看出，一个转换器至少应该实现两种类型的转换，一种是由 String 类型转换成指定数据类型；另一种是由指定数据类型转换成 String 类型。

注意：在将客户端提交数据由 String 类型转换成指定数据类型时，需要将参数 value 转换成 String 数组。原因是 Struts 2 在获取请求参数时利用 HttpServletRequest 的 getParameterMap()方法，该方法返回的 Map 对象的 value 是一个 String 数组。

Struts 2 提供了一个抽象类 org.apache.Struts 2.model.StrutsTypeConverter，该类是从 ognl.DefaultTypeConverter 扩展而来。StrutsTypeConverter 具有两个抽象方法，用于实现 String 和其他数据类型之间的转换。

（1）public abstract Object convertFromString(Map context, String[] values, Class toClass)：该方法用于将一个或多个 String 转换成指定数据类型。参数 context 表示转换时所处的上下文的 Map 对象，参数 values 是要转换的字符串，参数 toClass 为要转换的目标类型。

（2）public abstract String convertToString(Map context, Object o)：该方法用于将指定数据类型转换成 String 类型。参数 context 表示转换时所处的上下文的 Map 对象，参数 o 是要转换的对象。

下面编写一个用于处理复数的程序。当客户端提交一个复数形式的字符串时，转换器能够将其自动换成复数形式；当服务器向客户端输出一个复数时，转换器能够将其转换成字符串形式。

代码 11-12 所示为一个复数类 Complex。

代码 11-12 Complex 类

```
package org.ch11.model;
public class Complex {
    private float realPart;
    private float imagPart;
    //省略了 realPart 和 imagPart 的 getter 和 setter 方法
    public String toString(){
        String complex;
        if(realPart!=0&&imagPart>0)           //实部不为 0,并且虚部大于 0
            complex=realPart+"+"+imagPart+"i";
        else
        {if(realPart!=0&&imagPart<0)          //实部不为 0,并且虚部小于 0
             complex=realPart+""+imagPart+"i";
         else
            if(imagPart==0)                  //虚部等于 0
                complex=realPart+"";
            else                             //实部等于 0
                complex=imagPart+"i";
        }
        return complex;
    }
}
```

代码 11-13 定义了一个 ComplexAction 类，该类包含三个 Complex 类型的属性 c1、c2 和 result。execute()方法实现了两个 Complex 类型数据的加法。

代码 11-13 ComplexAction.java

```
package org.ch11.action;

import java.model.Map;
import org.ch11.model.Complex;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class ComplexAction extends ActionSupport {
    private Complex c1;
    private Complex c2;
    private Complex result;
    //省略了 c1、c2 和 result 的 getter 和 setter 方法
    public String execute() throws Exception {
        result=new Complex();
        result.setImagPart(c1.getImagPart()+c2.getImagPart());
        result.setRealPart(c1.getRealPart()+c2.getRealPart());
        return SUCCESS;
    }
}
```

代码 11-14 给出了用于测试 ComplexAction 类的页面 complex.jsp。该页面既是用户输入两个复数的页面，也是 ComplexAction 类的结果页面。

代码 11-14　complex.jsp

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="/struts-tags" prefix="s" %>
<html>
    <head><title>复数</title></head>
    <body>
        <s:form action="add.action">
            <s:textfield name="c1" label="复数 1"></s:textfield>
            <s:textfield name="c2" label="复数 2"></s:textfield>
            <s:submit></s:submit>
        </s:form>
        结果:<s:property value="result"/><br/>
    </body>
</html>
```

代码 11-15 为 Complex 类型和字符串之间的转换器类 ComplexConverter。该类扩展自 StrutsTypeConverter。

代码 11-15　ComplexConverter.java

```
package org.ch11.converter;

import java.model.Map;
import org.apache.struts2.model.StrutsTypeConverter;
import org.ch11.model.Complex;
public class ComplexConverter extends StrutsTypeConverter {
    //将客户端输入的字符串转换成 Complex 类型
    public Object convertFromString(Map context, String[] o,Class toClass){
        Complex c=new Complex();
        String s=o[0];
        int index=s.lastIndexOf('i');              //获得虚部 i 的位置

        if(index==-1){                             //s 为实数
            c.setRealPart(Float.parseFloat(s));
            c.setImagPart(0);
        }
        else                                       //s 为虚数
        {
            int i;
            for(i=s.length()-2;i>=0;i--)           //查找虚部的符号位置
             if(s.charAt(i)=='-'||s.charAt(i)=='+')
                break;
            if(i==-1)                              //纯虚数,且虚部为正数
                i=0;

            String img=s.substring(i, index);
```

```
            c.setImagPart(Float.parseFloat(img));

            if(i!=0)                            //实部和虚部均不为 0
                c.setRealPart(Float.parseFloat(s.substring(0,i)));
            else                                //纯虚数
                c.setRealPart(0);
        }
        return c;
    }
    //将 complex 类型转换成字符串类型
    public String convertToString(Map context, Object o) {
        Complex c= (Complex)o;
        return c.toString();
    }
}
```

11.3.2 配置类型转换器

在编写完自定义的类型转换器之后，还要将该类型转换器进行配置，才能在 Web 应用程序中使用。程序员根据需要，可以将类型转换器配置成局部类型转换器和全局类型转换器两种。

1. 局部类型转换器

局部类型转换器需要为 Action 的每一个属性分别指定类型转换器。局部类型转换器需要提供如下格式的文件。

```
ActionName- conversion.properties
```

其中，ActionName 为 Action 类的名字。该 Action 类的某个或某些属性需要利用自定义类型转换器进行类型转换。-conversion. properties 是固定不变的格式。

类型转换配置文件是一个典型的属性文件，该文件内容的形式如下：

```
field1= customConverter1
field2= customConverter2
...
```

例如，为上面的 ComplexAction 类提供的类型转换配置文件的名字为 ComplexAction-conversion. properties，其内容如代码 11-16 所示。

代码 11-16 ComplexAction-conversion. properties

```
c1= org.ch11.converter.ComplexConverter
c2= org.ch11.converter.ComplexConverter
result= org.ch11.converter.ComplexConverter
```

上述配置文件分别为 ComplexAction 类的属性 c1、c2 和 result 配置了自定义类型转换器。

最后，需要提醒大家的是，该类型转换配置文件一定要和 Action 类放到同一个子目录中。

2 配置全局类型转换器

局部类型转换器仅对某个 Action 类中的属性有效，如果所有 Action 类的特定类型的属性都具备相同的类型转换方式，可以为该类型配置全局的类型转换器。全局类型转换器的名字为 xwork-conversion. properties，位于 Web 应用程序的 WEB-INF/classes/子目录（MyEclipse 工程的 src 目录）下，文件内容如下：

```
fullyQualifiedClassName1=CustomerConverter1
...
```

例如，为前面定义的 Complex 类型数据定义的全局类型转换配置文件如代码 11-17 所示。

代码 11-17 xwork-conversion. properties

```
org.ch11.model.Complex=org.ch11.converter.ComplexConverter
```

此时，无论在工程的哪个类中用到 Complex 类型的数据，都将使用 ComplexConverter 转换器进行类型转换。

在浏览器中浏览 complex. jsp，运行结果如图 11-4 所示。

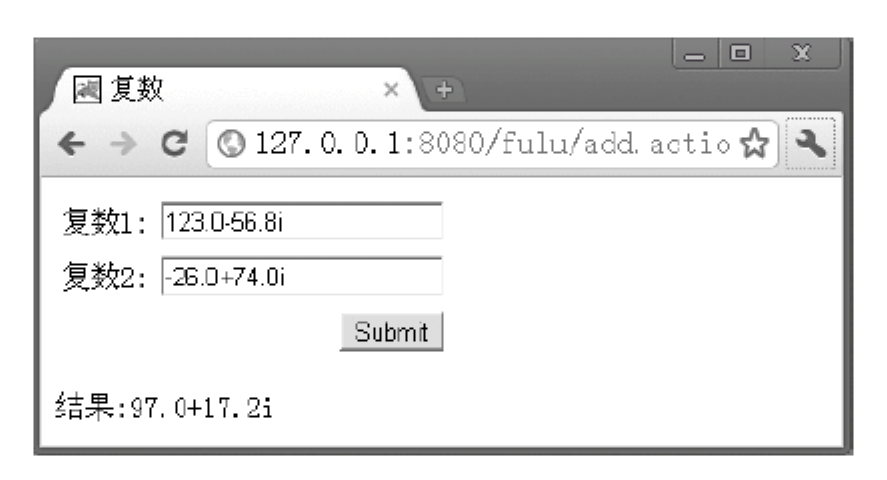


图 11-4 复数类型的转换

11.4 类型转换中的错误处理

用户在填写表单中的数据时，经常会输入一些格式不正确的数据。例如，在图 11-4 所示的页面中，将复数 1 的值输入为“123. 0—56. 8i”。当输入数据格式不正确时，在类型转换过程中会导致错误的发生。Conversion Error 拦截器将产生错误的字段和错误信息放到 ActionContext 中名为 conversionErrors 的 Map 对象中，当表单和发生错误的字段采用的不是 simple 主题时，该字段将输出一条 field 级错误信息。

```
Invalid field value for field "c2"
```

这里，c2 为产生错误的字段名。

上述错误提示是 xwork 默认的输入错误消息，定义在 xwork-messages. properties 文件中。如果希望为用户提供一条中文的错误消息提示，可以为应用程序配置一个资源

文件,并配置如下资源信息。

```
xwork.default.invalid.fieldvalue=数据类型不合法
```

具体配置方法可以参见第 10 章国际化资源文件的配置部分。

一旦资源 xwork.default.invalid.fieldvalue 被覆盖,应用程序中的所有类型转换的错误提示都将变成"数据类型不合法"。如果希望为不同的字段提供不同的错误提示,可以为 Action 类提供一个 Action 范围的资源文件,并编写如下格式的资源消息。

```
invalid.fieldvalue.fieldname=message
```

其中,invalid.fieldvalue 是固定的写法;fieldname 是需要定制错误消息的 Action 类的属性;message 为错误提示消息。

代码 11-18 给出了为 ComplexAction 类的 c1 和 c2 属性定制错误消息的资源文件。

代码 11-18　ComplexAction.properties

```
invalid.fieldvalue.c1=c1 不是有效的复数!
invalid.fieldvalue.c2=c2 不是有效的复数!
```

除了可以定制错误消息的文字外,还可以定制错误消息的 CSS 样式。JSP 在输出错误消息时,会将错误消息放在一个 HTML 的 span 标签中,并将它的 class 属性指定为 errorMessage。因此,程序员可以通过覆盖 errorMessage 来改变错误消息的输出样式。例如,下面的代码将错误信息的颜色设置为红色。

```
<style type="text/css">
    .errorMessage{color:red}
</style>
```

图 11-5 给出的 complex.jsp 显示了定制后的错误提示。



图 11-5　显示定制了错误的 complex.jsp

同步训练

1. 编写 Web 程序,实现格式为(x,y,z)形式的空间坐标的输入。编写空间坐标类、坐标类型转换器、转换配置文件以及相应的 Action 类和输入/输出页面。

2. 编写 Web 程序,实现一组空间坐标的输入,要求分别使用 List 和 Set 实现。

Chapter 12

第12章 注 解

12.1 注解概述

在前面的章节中，通常在 struts.xml 中进行应用程序的配置。事实上，Struts 2 还提供了一种称为注解（Annotation）的技术，允许程序员在不使用 struts.xml 的情况下，完成对 Action、Result、package 和拦截器等的配置。

从 Struts 2.1 版本开始，Struts 2 使用新的注解插件 convention-plugin 代替了以往的 codebehind-plugin。与 codebehind-plugin 相比，convention-plugin 有如下特点。

（1）通过包命名习惯来确定 Action 的位置。

（2）用命名习惯来确定 Result（支持 JSP、FreeMarker 等）映射路径。

（3）通过类名约定结果视图的 URL。

（4）通过包名约定命名空间的名称。

（5）采用遵循 SEO 规范的链接地址，例如用 hello-action 替代 HelloAction。

（6）使用注解配置 Action 的名字。

（7）使用注解配置 Action 的拦截器。

（8）使用注解配置命名空间。

（9）使用注解配置 XWork 包。

（10）提供默认 Action 以及默认的结果。例如，对于客户请求/products，convention-plugin 尝试寻找 com.example.actions.Products 或 com.example.actions.products.Index 进行处理。

convention-plugin 插件允许在程序员没有进行任何配置的情况下，通过约定完成自动配置。

如果要利用 convention-plugin 注解插件完成应用程序的配置，首先应该确保将以下三个架包引入到工程中。

```
asm-x.x.jar
asm-commons-x.x.jar
struts2-convention-plugin-xxx.jar
```

这三个 jar 文件可以从 Struts 2 下载包中找到。

需要注意的是，使用 convention-plugin 对应用程序的行为进行注解，并不能完全抛弃 struts.xml。注解只是减少了程序员在 struts.xml 中进行与 Action、Result 和拦截器

有关的配置。

12.2 约定

假设在 Web 应用程序的 WEB-INF/content 目录下存放着一个 hello.jsp 页面，当通过浏览器访问 http://localhost:8080/hello 或 http://localhost:8080/hello.action 时，能够看到浏览器中显示出 hello.jsp 的内容。令人惊讶的是用户根本就没有编写 hello.action 对应的 Action 类。所有这些工作都是由 convention-plugin 自动完成的。

下面介绍 convention-plugin 默认时使用的一些约定。

1. 结果视图存储路径的约定

默认时，convention-plugin 假设所有的结果页面存放在 WEB-INF/content 目录下。

如果用户希望修改结果页面存放目录，可以在 struts.xml 或 struts.properties 中重新配置。例如，在 struts.xml 配置

```
<constant name="struts.convention.result.path" value="/WEB-INF/jsp" />
```

所有结果页面的存放路径将被修改为/WEB-INF/page。

2. Action 类存放路径的约定

convention-plugin 是如何找到 Web 应用程序中的 Action 类的呢？在默认时，所有名称中具有 action、actions、struts 和 Struts 2 的包都会被 convention-plugin 作为含有 Action 类的路径来搜索。在这些包的路径下，所有 com.opensymphony.xwork2.Action 的实现类以及以名字使用 Action 结尾的类都是 convention-plugin 所要处理的 Action 类。

如果用户希望自定义 convention-plugin 搜索的包名，可以在 struts.properties 或 struts.xml 中修改 struts.convention.package.locators 属性。例如：

```
<constant name="struts.convention.package.locators" value="Action,Actions" />
```

这时 convention-plugin 只在名称含有 Action 和 Actions 的包中搜索 Action 类。

3. 命名空间的约定

convention-plugin 约定，从定义的 package.locators 开始到包结束的部分，就是命名空间。例如：

（1）com.example.actions.MainAction 的命名空间为“/”。

（2）com.example.actions.products.Display 的命名空间为/products。

（3）com.example.struts.company.details.ShowCompanyDetailsAction 的命名空间为/company/details。

convention-plugin 利用类名确定资源的 URL。首先，它将类名中的 Action 去掉；如果发现类名采用的是驼峰式的命名规则（英文单词首字母大写），会将所有驼峰字母转换成小写，并使用“-”连接。例如：

（1）com.example.actions.MainAction 对应的 URL 为/main。

(2) com. example. actions. products. Display 对应的 URL 为/products/display。

(3) com. example. struts. company. details. ShowCompanyDetailsAction 对应的 URL 为/company/details/show-company-details。

注意：com. example. actions. helloWorldAction 对应的 URL 为/helloWorld，而不是 hello-world。

4. Result 和 Results 的约定

convention-plugin 约定如下。

(1) 当要访问的资源没有对应的 Action 存在，访问资源的名称就直接和页面的名称对应。必须要注意的是，访问时用到的资源名称和页面的名字必须是小写的。这就是为什么在前面的例子中，在没有进行任何配置的情况下，用户也能访问到 hello. jsp 页面。

(2) 在存在对应的 Action 的情况下，优先使用“Action 的 URL＋Result 的字符串＋文件类型的后缀”的方式查找对应的结果页面。当没有找到时，将按照“Action 的 URL＋文件类型的后缀”重新查找。

表 12-1 中给出了几个 result 结果映射的示例。

表 12-1 result 结果映射的示例

URL	Result	能够匹配的文件	Result 类型
/hello	success	/WEB-INF/content/hello. jsp	Dispatcher
/hello	success	/WEB-INF/content/hello-success. htm	Dispatcher
/hello	success	/WEB-INF/content/hello. ftl	FreeMarker
/hello-world	input	/WEB-INF/content/hello-world-input. vm	Velocity
/test1/test2/hello	error	/WEB-INF/content/test/test2/hello-error. html	Dispatcher

5. Action 链的约定

如果一个 Action 返回的结果码是同一个 package 内另外一个 Action 的名字，并且第一个 Action 返回的 result 没有对应的页面存在，这两个 Action 将形成一个 Action 链。

12.3 利用注解代替 struts.xml

12.3.1 @Action 和@Actions

Convention 除了约定利用类名确定资源的 URL 外，还提供了一种名为@Action 的注解，允许用户将 Action 类映射成其他的 URL。@Action 注解包含以下属性。

(1) value：指定所要映射的 URL 的名称。

(2) params：指定需要注入 Action 的参数。

(3) results：指定结果视图。

(4) interceptorRefs：指定拦截器引用。

(5) excepitonMappings：声明错误处理。

params 属性的格式如下：

params={"参数 1","第 1个参数的值","参数 2","第 2个参数的值",…,"参数 *n*","第*n*个参数的值"}

当需要将同一个 Action 类映射为多个 URL 时,需要将这些@Action 放到一个@Actions 中。

来看一个例子,如代码 12-1 所示。

代码 12-1 @Action 注解

```
package com.zero.action;
import org.apache.struts2.convention.annotation.* ;
import com.opensymphony.xwork2.ActionSupport;

public class actionAnno extends ActionSupport {
    private int param;
    private String message;
    //省略了 get 和 set 方法

    @Actions({
        @Action(value="/action1",
                params={"param","-50","message","Good Morning!"}
        ),
        @Action(value="/action2",
                params={"param","100","message","hello wolrd!"}
        )
    })
    public String execute(){
        return SUCCESS;
    }

    @Action("url")
    public String good() {
        return SUCCESS;
    }
}
```

上述代码通过@Actions 和@Action 将 actionAnno 类的 execute()方法的 URL 分别映射成/action1 和/action2,并为参数 param 和 message 注入不同的值。Good()方法的 URL 地址是不推荐的,上面 url 将使用 java 包名作为 namespace,而不会直接使用 Action 注解的地址。

注意:

(1) 一般情况下,尽可能不将同一个 Action 类映射成不同的 URL,以免发生混淆。

(2) 一旦某个 Action 类使用了@Action 注解,原来利用类名约定资源的 URL 将不再对该类有效,以下其他注解也是同样的。例如,上例中已经不能再利用/actionAnno 去访问该 Action 类了。

12.3.2 @Result 和@Results

Convention plugin 允许将 Action 的结果映射到约定之外的其他视图上。

在 Action 类层面上定义的@Result 和@Results 是全局的注解，将被该 Action 类中定义的所有 Action 所共享。在 method 层面上定义的@Result 和@Results 是局部的注解，仅应用于当前的 method。当全局注解和局部注解冲突时，起作用的是局部注解。

@Result 注解用于声明一个结果，有以下属性。

（1）name：指定结果的名字，默认为 success。

（2）type：指定结果的类型，默认为在<result-types>中定义的默认类型。

（3）location：指定结果的页面位置。

（4）params：指定结果的可选参数。

当一个 Action 有多个结果视图页面时，需要利用@Results 注解将多个@Result 组合起来。@Results 只有一个属性 value，其值为一个@Result 数组。

下面介绍一个在 Action 类中使用@Result 和@Results 注解的例子，如代码 12-2 所示。

代码 12-2 @Result 和@Results 注解

```
package com.zero.action;
import org.apache.struts2.convention.annotation.* ;
import com.opensymphony.xwork2.ActionSupport;

@Results({
    @Result(name="error", location="error.jsp")
    })
public class resultAnno extends ActionSupport {
    private int param;
    private String color;
    /* 省略了 get 和 set 方法 * /

    @Action(value="/result",
        results={
            @Result(location="hello.jsp"),
            @Result(name="input",
                location="hello-input.jsp", type="redirect")}
    )
    public String execute(){
        if(param==1){
            color="red";
            return SUCCESS;
        }else if(param==2){
            color="green";
            return INPUT;
        }
        return ERROR;
    }
```

```
        public String good() {
            System.out.println("do something");
            return SUCCESS;
        }
    }
```

假设在修改 struts.convention.result.path 的情况下：

(1) 访问 http://127.0.0.1:8080/zero/result?param=1，将转到/WEB-INF/content/hello.jsp 页面。

(2) 访问 http://127.0.0.1:8080/zero/result?param=2，将转到/index.jsp 页面。注意，当@Result 的 type 被指定为“redirect”时，location 中的结果视图将不再默认为相对于 struts.convention.result.path 所设置的目录，而是相对于应用程序的根目录。

(3) 访问 http://127.0.0.1:8080/zero/result! good?param=3，结果视图将变为/WEB-INF/content/error.jsp。

12.3.3 @Namespace

如果不希望 Convention plugin 使用约定的包名作为命名空间，可以使用@Namesapce 注解修改 Action 类的 namespace。@Namesapce 注解可以放在 Action 类中，或者是放在一个独立的名为 package-info.java 的文件中。在 Action 中编写的@Namespace 注解会作用在该 Action 中所有的相对 URL 上。在 package-info.java 文件中编写的@Namespace 注解会作用在位于该包中的所有的 Action（不包括子包中的 Action）。

@Namespace 注解只有 value 属性，用来指明包命名空间。

下面介绍一个在 Action 类中使用@Namespace 注解的例子，如代码 12-3 所示。

代码 12-3 @Namespace 注解

```
package com.zero.action;
import org.apache.struts2.convention.annotation.*;
import com.opensymphony.xwork2.ActionSupport;

@Namespace("/custom")
public class namespaceAnno extends ActionSupport {
    @Action("/sub/hello")
    public String execute(){
        return SUCCESS;
    }

    public String good() {
        System.out.println("do something");
        return SUCCESS;
    }
}
```

上例中可以使用/cutom/hello 和/sub/hello 两个不同的 URL 去访问 Action。

接下来介绍如何在 package-info.java 中配置@Namespace 注解，如代码 12-4 所示。

代码 12-4 在 package-info.java 中配置@Namespace 注解

```
@org.apache.struts2.convention.annotation.Namespace("/custom")
package com.zero.actions;
```

这样，包 package com.zero.actions 中所有 Action 的默认命名空间被修改成/custom。

@Namesapce 注解用于更改 Action 类的 namespace。这个注释用于一个 Action 类或者用于 package-info.java。当用于 Action 类，将影响该类中定义的所有 Action；当用于 package-info.java，将影响该包中所有的 Action，但不影响该包的子包中的 Action。

12.3.4 @ResultPath 注解

@ResultPath 注解用来改变结果页面所在的目录，参见代码 12-5。

代码 12-5 @ResultPath 注解

```
package com.zero.action;
import org.apache.struts2.convention.annotation.*;
import com.opensymphony.xwork2.ActionSupport;
@ResultPath("/WEB-INF/page")
public class resultPathAnno extends ActionSupport {

    public String execute(){
        return SUCCESS;
    }

    @Action("myurl")
    public String good() {
        System.out.println("do something");
        return SUCCESS;
    }
}
```

12.3.5 @ParentPackage

@ParentPackage 注解用来为 Action 指定父包，它只有一个 value 属性，用来指定父包的名字。

下面介绍一个在 Action 类中使用@ParentPackage 注解的例子，如代码 12-6 所示。

代码 12-6 @ParentPackage 注解

```
package com.zero.actions;
import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.convention.annotation.Action;
```

```
import org.apache.struts2.convention.annotation.ParentPackage;

@ParentPackage("customXWorkPackage")
public class packageAction extends ActionSupport {
    public String execute() {
        return SUCCESS;
    }
}
```

如果希望为所有的 Action 指定父包，可以在 struts.xml 中修改属性 struts.convention.default.parent.package 的值。

12.3.6 @InterceptorRef 和@InterceptorRefs 注解

@Interceptor 注解用来为 Action 引用拦截器，它只有一个 value 属性，用来指定引用的拦截器或拦截器栈的名字。

当需要为 Action 配置多个@Interceptor 注解时，可以将这些@Interceptor 注解组成一个数组，然后赋值给@InterceptorRefs 注解的 value 属性。

下面介绍一个在 Action 类中使用@InterceptorRef 和@InterceptorRefs 注解的例子，如代码 12-7 所示。

代码 12-7 @InterceptorRef 和@InterceptorRefs 注解

```
package com.zero.action;
import org.apache.struts2.convention.annotation.* ;
import com.opensymphony.xwork2.ActionSupport;

@ParentPackage("auth-Package")
@InterceptorRefs({
    @InterceptorRef("auth")
    ,@InterceptorRef("defaultStack")
})
public class intercepterAnno extends ActionSupport {
     @Action(value="ip", interceptorRefs=@InterceptorRef("validation")
            ,results={@Result(name="login",location="/login.jsp")
        ,@Result(name="success",location="/index.jsp")
    }
    )

    public String execute(){
        return SUCCESS;
    }
}
```

12.3.7 @ExceptionMapping 和@ExceptionMappings 注解

@ExceptionMapping 注解用来为 Action 配置异常处理。与@result 注解类似，该注

解也可以使用 params 属性配置要传入的参数。另外,该注解可以为整个 Action 类配置配置异常处理,也可以为某一个 method 配置异常处理。

@ExceptionMapping 注解有以下属性。

(1) exception:用于指定需要捕获的异常。

(2) result:用于指定捕获异常之后跳转到的结果视图。

(3) params:用于指定参数列表。

当需要声明多个错误处理时,必须为这些错误分别编写@ExceptionMapping 注解,然后将其组成数组,放到一个@ExceptionMappings 注解中。

下面介绍一个在 Action 类中使用@ExceptionMapping 和@ExceptionMappings 注解的例子,如代码 12-8 所示。

代码 12-8 @ExceptionMapping 和@ExceptionMappings 注解

```
@ExceptionMappings({
    @ExceptionMapping(
        exception= "java.lang.NullPointerException"
        ,result= "success"
        ,params= {"param1", "val1"})
})
public class ExceptionsActionLevelAction {
    public String execute() throws Exception {
        return null;
    }
}
```

124 案例 8: 利用注解配置留言板程序

本节通过实例介绍如何利用注解取代 struts. xml 中与 Action 配置、Result 映射和拦截器等有关的配置。

1. 准备工作

首先,在 MyEclipse 集成环境中新建一个 Web Project 工程,取名为 zNotes。接下来,将留言板程序工程中的源码、静态资源(页面、图片和样式表等)、web. xml 和 jar 库复制到 zNotes 工程中。在实际操作时,可以直接将 src 目录和 WEB-INF 目录及其他资源目录直接复制到 zNotes 中。

将 asm-x. x. jar、asm-commons-x. x. jar 和 Struts 2-convention-plugin-xxx. jar 引入工程。

2. 配置 struts.xml

使用注解配置应用程序并不能完全取代 struts. xml,一些常量的配置和拦截器的配置仍然需要在 struts. xml 中完成。修改 zNotes 的 struts. xml,如代码 12-9 所示。

代码 12-9 struts. xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<constant name="struts.convention.result.path" value="/WEB-INF/jsp"/>

    <package name="zero-notes" namespace="/" extends="convention-default">
        <interceptors>
            <interceptor name="security"
                class="notes.interceptor.AuthenInterceptor"></interceptor>
            <interceptor-stack name="auth">
                <interceptor-ref name="defaultStack"></interceptor-ref>
                <interceptor-ref name="security"></interceptor-ref>
            </interceptor-stack>
        </interceptors>
    </package>
</struts>
```

由于将结果视图放在了/WEB-INF/jsp 目录下，而不是默认的/WEB-INF/content 下面，因此一定要配置常量 struts. convention. result. path。

代码 12-9 中还配置了一个名为 zero-notes 的 package，它是从 convention-default 上扩展而来的。注意 package 的名字，在后面为 Action 类添加@InterceptorRef 注解时要用到它。在 package 中，还配置了一个用于登录验证的拦截器。在使用注解配置应用程序时，拦截器的配置是在 struts. xml 中的。在 Action 类中，@InterceptorRef 和@InterceptorRefs 注解将引用这些拦截器。

3. 配置 LoginAction 类

代码 12-10 给出了利用 Convention plugin 注解过的 LoginAction 类。

代码 12-10 添加了注解的 LoginAction 类

```
package notes.action;
import java.util.Map;
import org.apache.struts2.convention.annotation.* ;
import notes.dao.UserDao;
import notes.dao.impl.UserDaoImpl;
import notes.model.User;
import com.opensymphony.xwork2.* ;

public class LoginAction extends ActionSupport {
    private String userName;
    private String password;
    //省略了 get 和 set 方法

    @Action(value="login"
```

```
            ,results= {@Result(name= "login",location= "login.jsp")
            ,@Result(name= "success",location= "listNotes",type= "chain")}
    )
    public String execute() throws Exception {
        UserDao userDao= new UserDaoImpl();
        User user= userDao.findUser(userName, password);

        if(null= = user){
            this.addActionError("用户名或密码错误!");
            return LOGIN;
        }
        else{
            ActionContext context= ActionContext.getContext();
            Map session= context.getSession();
            session.put("user", user);
            return SUCCESS;
        }
    }

    @Action(value= "logout"
            ,results= {@Result(name= "login",location= "login.jsp")}
    )
    public String logout(){
        ActionContext context= ActionContext.getContext();
        Map session= context.getSession();
        session.clear();
        return LOGIN;
    }
}
```

上述代码分别为 LoginAction 的 execute()方法和 logout()方法添加了@Action 注解,并利用@Result 对每种结果码注解了对应的结果视图。注意,Action 之间的 chain 链的配置方法。

4. 配置 NotesAction 类

代码 12-11 给出了利用 Convention plugin 注解过的 NotesAction 类。

代码 12-11　添加了注解的 NotesAction 类

```
package notes.action;

import java.io.* ;
import java.util.* ;
import notes.dao.NotesDao;
import notes.dao.impl.NotesDaoImpl;
import notes.model.* ;
import org.apache.struts2.ServletActionContext;
import org.apache.struts2.convention.annotation.* ;
import com.opensymphony.xwork2.* ;
```

```
@ParentPackage("zero-notes")
@InterceptorRef("auth")
public class NotesAction extends ActionSupport {
    private Notes note;
    private List<Notes> notes=new ArrayList<Notes>();
    private String uploadDir;
    private int noteId;
    private File attachment;
    private String attachmentFileName;
    private String attachmentContentType;
    //省略了 get 和 set 方法

    @Action(value="listNotes", interceptorRefs=@InterceptorRef("auth")
            ,results={@Result(name="login",location="login.jsp")
                    ,@Result(name="success",location="index.jsp")
            }
    )
    public String list() {
        NotesDao notesDao=new NotesDaoImpl();
        notes=notesDao.getAllNotes();
        return SUCCESS;
    }

    @Action(value="addNote", interceptorRefs=@InterceptorRef("auth")
        ,results={@Result(name="login",location="login.jsp")
                ,@Result(name="input",location="post.jsp")
                ,@Result(name="success",location="listNotes",type="chain")
        }
        ,params={"uploadDir","uploadFiles"}
    )
    public String add() {
        if (null==note)
            return INPUT;
        ActionContext context=ActionContext.getContext();
        Map session=context.getSession();
        User user=(User) session.get("user");
        if (null==user)
            return LOGIN;
        note.setUser(user);
        if(null!=attachment)
            upload();
        NotesDao notesDao=new NotesDaoImpl();
        notesDao.addNote(note);
        return SUCCESS;
    }

    @Action(value="detail", interceptorRefs=@InterceptorRef("auth")
        ,results={@Result(name="login",location="login.jsp")
```

```
                    ,@Result(name="success",location="detail.jsp")
            }
            ,params={"uploadDir","uploadFiles"}
        )
        public String detail() {
            NotesDao notesDao=new NotesDaoImpl();
            note=notesDao.getNoteById(noteId);
            return SUCCESS;
        }

        private void upload(){
            /* 代码略,请参照 Struts 文件上传一章的案例 */
        }
    }
```

上述代码首先利用注解@ParentPackage("zero-notes")声明了 NotesAction 类中所要引用的拦截器所在的包(见 struts. xml 中配置),然后利用注解@InterceptorRef("auth")将所要用到的拦截器引入类中,最后利用注解@InterceptorRef("auth")为每个方法声明了拦截器引用。由于 auth 拦截器在验证失败时返回“login”结果码,所以为每一个方法添加了一个名为 login 的@Result 注解。

5. 配置 FileDownloadAction 类

代码 12-12 给出了利用 Convention plugin 注解过的 FileDownloadAction 类。

代码 12-12　添加了注解的 FileDownloadAction 类

```
package notes.action;
import java.io.*;
import javax.servlet.ServletContext;
import org.apache.struts2.ServletActionContext;
import org.apache.struts2.convention.annotation.*
import com.opensymphony.xwork2.ActionSupport;

@ParentPackage("zero-notes")
@InterceptorRef("auth")
public class FileDownloadAction extends ActionSupport {
    private String fileName;        //初始的通过 param 指定的文件名属性
    private String downloadDir;     //下载文件所在的目录
    //省略了 get 和 set 方法

    public String getContentDisposition(){
        String s;
        s="attachment;filename=\""+getDownloadFileName()+"\"";
        return s;
    }

    public InputStream getInputStream() throws Exception {
        /* 代码略,请参照 Struts 文件上传一章的案例 */
```

```
    }

    @Action(value="download", interceptorRefs=@InterceptorRef("auth")
            ,results={@Result(name="login",location="login.jsp")
                    ,@Result(name="success",type="stream")
             }
            ,params={"downloadDir","uploadFiles","bufferSize","4096"}
    )
    public String execute() throws Exception {
        return SUCCESS;
    }

    public String getDownloadFileName() {
        /* 代码略,请参照 Struts 文件上传一章的案例 */
    }
}
```

读者可以参照 NotesAction 类中的注解分析,自行分析上述代码中的注解。

同 步 训 练

删除 struts.xml 中与 Action 配置、Result 映射和拦截器引用有关的内容,然后利用注解配置站内短信系统。

Chapter 13

第13章 整合 JQuery

13.1 JQuery 语法

13.1.1 JQuery 简介

JQuery 是目前最为流行的 JavaScript 框架之一，它允许程序员使用现有的知识，例如 CSS、HTML、XHTML 以及原生的 JavaScript 直接操作页面元素，无须学习 JavaScript 高级特性，实现快速开发。

JQuery 强调写得少，做得多(write less, do more)。JQuery 提供了强大的选择器，允许使用 CSS1 至 CSS3 中提供的几乎所有的选择器，还自定义了一些选择器。另外，JQuery 封装了大量与 DOM 有关的操作，使得程序员能够轻松完成各种复杂的操作。JQuery 利用 Ajax()函数封装了所有的 AJAX 操作，使得程序员在开发 AJAX 操作时无需关心浏览器兼容问题以及 XMLHttpRequest 对象的创建和使用问题。利用 JQuery 编写的代码能够在所有主流浏览器上顺畅运行。JQuery 修复了浏览器之间的差异。由于 JQuery 的易扩展性，引发了全球各地的开发者编写 JQuery 的扩展插件，程序员可以利用这些插件实现各种 JavaScript 操作和特效。

在编写本书时，JQuery 的最新版本是 1.8.1。读者可以从 http://jquery.com/下载 JQuery 的最新版本，并将其复制到自己的工程中。

13.1.2 JQuery 选择器

JQuery 中的选择器继承了 CSS 的风格。利用 JQuery 的选择器能够方便、快捷地找到特定的 DOM 元素，以便对它们的属性进行操作，或为它们添加新的行为。

JQuery 选择器分为基本选择器、层次选择器、过滤选择器和表单选择器。

基本选择器是 JQuery 中最常用的选择器，它利用元素的 id、class 和标签名字来选择 DOM 元素。表 13-1 给出了 JQuery 中用到的基本选择器。

表 13-1 基本选择器

选择器	说明	举例
#id	根据给定的 id 选择元素	$("#intro")选择 id="intro"的第一个元素
.class	根据给定的 class 选择元素	$(".intro")选择所有 class="intro" 的元素
element	选择给定元素名的所有元素	$("p") 选择所有 <p> 元素

续表

选择器	说明	举例
*	选择所有元素	$("*")
this	选择当前 HTML 元素	$("this")
selector1,selector2,…,selectorN	将多个选择器组合成一个	$("p,span ")选择所有的<p>和<span> 元素

表 13-1 中的 $()函数是 JQuery 的选择器函数,用于选择页面中的元素。

层次选择器允许 JQuery 根据 DOM 之间的层次关系选择特定的元素,例如兄弟元素和后代元素等。表 13-2 给出了 JQuery 用到的层次选择器。

表 13-2 层次选择器

选择器	说明	举例
$("ancestor descendant")	选择 parent 元素后所有的 child 元素	$("#test div") 选取 id 为 test 的元素所包含的所有的 div 子元素
$("parent > child")	选择 parent 元素后所有的第一级 child 元素	$("#div_a1 > input")选择 id 为 div_a1 的 div 中的所有 input 元素
$("prev+next")	选择在 prev 元素后面的 next 元素	$("#demo+img")选在 id 为 demo 元素后面的 img 对象
$("prev～siblings")	选择 prev 后面的 siblings 元素	$("#demo～[title]")选择 id 为 demo 的元素后面所有带有 title 属性的元素

过滤选择器通过特定的规则选择所需的 DOM 元素。JQuery 中的过滤选择器分为基本过滤、内容过滤、可见性过滤、属性过滤、子元素过滤和表单对象属性过滤选择器。表 13-3 给出了 JQuery 中用到的基本过滤选择器。

表 13-3 基本过滤选择器

选择器	描述	示例
:first	选取第一个元素	$("tr:first")选取所有 tr 元素中的第一个 tr 元素
:last	选取最后一个元素	$("tr:last")选取所有 tr 元素中的最后一个 tr 元素
:even	选取索引是偶数的所有元素	$("tr:even")选取索引是偶数的 tr 元素
:odd	选取索引是奇数的所有元素	$("tr:odd")选取索引是奇数的 tr 元素
:eq(index)	选取索引等于 index 的元素	$("tr:eq(1) ")选取索引等于 1 的 tr 元素
:gt(index)	选取索引大于 index 的元素	$("tr:gt(1) ")选取索引大于 1 的 tr 元素
:lt(index)	选取索引小于 index 的元素	$("tr:lt(1) ")选取索引小于 1 的 tr 元素,等同于 $("tr:eq(0) ")
:header	选取所有的标题元素,例如 h1,h2,h3,…	$(":header")选取网页中所有的 h1,h2,h3,…
:animated	选取当前正在执行动画的所有元素	$("div:animated")选取正在执行动画的 div 元素

注意:索引是从 0 开始的。

表单选择器是用于获取表单的某个或某类型的元素。表单选择器都是以冒号(":")开头,例如:

（1）$(":input")：选择表单中所有的<input>、<textarea>、<select>和<button>元素。

（2）$(":checkbox")：选择表单中所有的<checkbox>元素。

13.1.3　常用的 JQuery 属性方法

JQuery 提供的属性方法允许程序员访问一个元素所有的属性，包括 CSS 和样式属性，并且能够修改它们。常见的属性方法如表 13-4 所示。

表 13-4　常见的 JQuery 属性方法

方　法	描　　述	举　　例
addClass()	向匹配的元素添加指定的类名	$("p:first").addClass("intro")：向第一个 p 元素添加一个类
attr()	设置或返回匹配元素的属性和值	$("img").attr("width","180")：改变图像的 width 属性
hasClass()	检查匹配的元素是否拥有指定的类	$("p:first").hasClass("intro")：检查第一个<p>元素是否包含 "intro" 类
html()	设置或返回匹配的元素集合中的 HTML 内容	$("p").html("Hello world!")：设置所有 p 元素的内容 var o=$(".head").html()：获得 id 为 head 的元素的内容
removeAttr()	从所有匹配的元素中移除指定的属性	$("p").removeAttr("id")：从任何 p 元素中移除 id 属性
removeClass()	从所有匹配的元素中删除全部或者指定的类	$("p:first").removeClass("intro")：移除所有<p>的 intro 类
toggleClass()	从匹配的元素中添加或删除一个类	$("button").click(function(){ $("p").toggleClass("main"); })：对设置和移除所有<p>元素的 main 类进行切换
val()	设置或返回匹配元素的值	$(".tip").val("Hello World")：设置 id 为 tip 的元素的值 alert($(".tip").val())：获得 id 为 tip 的元素的值
hide()	隐藏被选的元素	$("p").hide()：隐藏可见的<p>元素
show()	显示被选的元素	$("p").show()：显示出隐藏的<p>元素

除上述方法外，JQuery 还提供了十几个方法可以为 DOM 元素添加动态效果，读者可以自行查阅 JQuery 手册。

13.1.4　常用的 JQuery 事件方法

许多脚本需要在特定事件(或浏览器动作)发生时进行特定的处理。

1. 文档加载事件方法

$(document).ready()是所有事件方法中最重要的一个。该方法在文档载入就绪时就对其进行操纵，并调用执行它所绑定的方法。

$(document).ready()的用法如下：

```
$(document).ready(function(){
```

```
            //事件代码
})
```

通常可以简写为

```
$(function(){
            //事件代码
})
```

2 为DOM元素绑定事件

JQuery中可以为DOM元素动态绑定的事件有blur、focus、focusin、focusout、load、resize、scroll、unload、click、dblclick、mousedown、mouseup、mousemove、mouseover、mouseout、mouseenter、mouseleave、change、select、submit、keydown、keypress、keyup和error等。每个事件方法的名字表明了事件方法的作用，在此不再赘述。

代码13-1演示了如何使用JQuery操作DOM对象。

代码13-1 JQuery综合应用

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>jquery example</title>

<style>
    .focus{border:1 solid #F00; background-color:#FCC;}
    .even{ background-color:#FFF38F}
    .odd{background-color:#FFFFEE;}
</style>
<script src="js/jquery-1.8.1.min.js"></script>
<script>
    $(function(){
        $(":input").focus(function(){      //当input控件获得焦点时,添加focus样式
            $(this).addClass("focus");
            }).blur(function(){            //当input控件失去焦点时,删除focus样式
                $(this).removeClass("focus");
            });

        $("#add").click(function(){        //为id=add的按钮添加click事件方法
            var data1=$("#data1").val();//获得id=data1输入框的值
            var data2=$("#data2").val();//获得id=data2输入框的值
            var result=parseInt(data1)+parseInt(data2)
            $("#result").val(result);      //将计算结果保存在id=result的输入框中
            var history=$("#history")      //获得id=history的table对象
            //以下代码用于向table表中动态添加一行数据
            var row=$("<tr></tr>");
            var td1=$("<td></td>");
            td1.text(data1)
            var td2=$("<td></td>");
```

```
                td2.text(data2)
                var td3=$("<td></td>");
                td3.text(result)

                row.append(td1);
                row.append(td2);
                row.append(result);
                history.append(row);

                $("tr:odd").addClass("odd");       //为表中的奇数行添加样式 odd
                $("tr:even").addClass("even");     //为表中的偶数行添加样式 even
            });
        })
</script>
</head>
<body>
<input type="text" id="data1"/>+
<input type="text" id="data2"/>
<input type="button" id="add" value="="/>
<input type="text" id="result" readonly/>
    <br/>
    <table style="width:300px; border:" id="history">
        <tr>
            <th>Data 1</th>
            <th>Data 2</th>
            <th>Result</th>
        </tr>
    </table>
</body>
</html>
```

代码 13-1 演示了利用 JQuery 完成两个数的加法运算，并将计算历史保存在一个 table 中。图 13-1 所示为上述代码在浏览器中的运行效果。

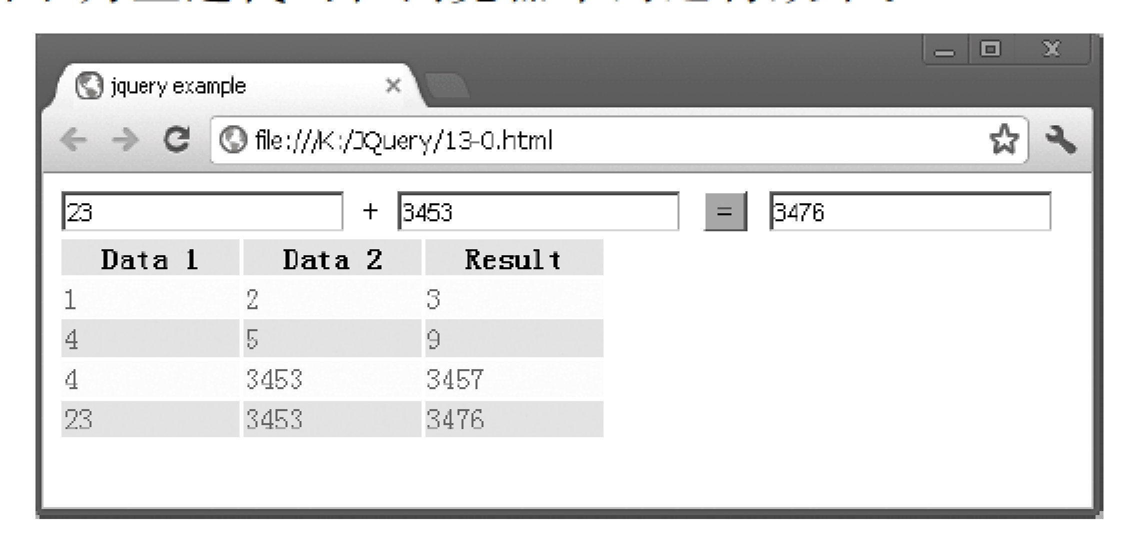


图 13-1　JQuery 方法演示

13.2 利用 JQuery 实现客户端验证

在 Web 开发时,JQuery 的一个重要作用就是验证用户在表单中的输入信息是否符合约束。例如,在前面的同步训练中,曾要求读者为留言板程序添加一个注册页面,并利用 Struts 2 提供的验证框架对用户信息进行验证。在实际的开发中,对于一些不会引发安全问题的输入信息的验证,会放到客户端来完成,这样可以大大减轻服务器的压力。因此,对于留言板注册页面的验证,可以利用 JQuery 来完成,如代码 13-2 所示。

代码 13-2 利用 JQuery 实现客户端验证

```
<%@page language="java" pageEncoding="GBK"%>
<%@taglib uri="/struts-tags" prefix="s"%>
<HTML>
<HEAD>
<TITLE>留言板--注册</TITLE>
<META http-equiv=Content-Type content="text/html; charset=gbk">
<link rel="stylesheet" type="text/css" href="style/main.css">
<script type="text/javascript" src="js/jquery-1.7.1.js"></script>
<script language="javascript">
$(function(){
    $("#submit").click(function(){
        var error=0;
        if($("#userName").val()==""){          /*验证用户名是否为空*/
            /*如果不存在 userTip 对象,添加一个 id 为 userTip 的 div*/
            if(!exist("userTip"))
                $("#userName").after("<div style='color:red'id='userTip'>用户名不能为空!</
                 div>");
            else                               /*存在 userTip 对象,直接显示*/
                $("#userTip").show();
                error++;
        }
        else
        $("#userTip").hide();

        if($("#password").val()==""){          /*验证密码是否为空*/
            if(!exist("passwordTip"))
                $("#password").after("<div style='color:red' id='passwordTip'>密码不能为空!</div
                 >");
            else
                $("#passwordTip").show();
            error++;
        }
        else
        $("#passwordTip").hide();
        /*验证两次密码是否一致*/
        if($("#password").val()!=$("#password2").val()){
          if(!exist("repasswordTip"))
```

```
                $("#password2").after("<div style='color:red' id='repasswordTip'>两次密码不一致!</div
                 >");
            else
                $("#repasswordTip").show();
            error++;
          }
          else
            $("#repasswordTip").hide();

          if($("#email").val()==""){                    /*验证 Email 是否为空*/
            if(!exist("emailTip"))
                $("#email").after("<div style='color:red' id='emailTip'></div>");
            else
                  $("#emailTip").show();
            $("#emailTip").text("Email 不能为空!");
            error++;
        }
        else{                                           /*验证 Email 是否符合规则*/
            var patten=new RegExp(/^[\w-]+(\.[\w-]+)*@([\w-]+\.)+[a-zA-Z]+$/);
            if(!patten.test($("#email").val())&&!exist("emailTip"))
              $("#email").after("<div style='color:red' id='emailTip'></div>");
            else
                $("#emailTip").show();
                $("#emailTip").text("不符合 Email 规则!");
                error++;
        }
        if(error>0)                           /*错误数超过 1 处,返回 false,禁止将请求发给服务端*/
            return false;
        })
})
/*判断是否存在某个 id值的 DOM元素*/
function exist(id){
        if($("#"+id).length>0){
            return true;
        }
        else
            return false;
}
</script>
</HEAD>
<BODY>
            <div class="header">
            </div>

<!--          用户注册表单         -->
<font color="red"><s:fielderror/></font>
        <s:form name="regForm" action="register.action" method="post" theme=
            "simple" namespace="/">
          <div class="col1" style="width:400px;">用户名</div>
```

```
        <div class="col2">
            <s:textfield name="userName" label="用 户 名" cssClass="input" id=
             "userName"></s:textfield>
        </div>
        <div class="clear"/>
        <div class="col1" style="width:400px;">密码</div>
        <div class="col2">
          <s:password name="password" label="密    码" cssClass="input" id=
           "password"></s:password>
        </div>
        <div class="clear"/>
        <div class="col1" style="width:400px;">重复密码</div>
        <div class="col2">
          <s:password name="password2" label="重 复 密 码" cssClass="input" id=
           "password2"></s:password>
        </div>
        <div class="clear"/>
        <div class="col1" style="width:400px;">Email</div>
        <div class="col2">
            <s:textfield name="email" id="email"></s:textfield>
        </div>
        <div class="clear"/>
        <div class="col1" style="width:400px;">性别</div>
        <div class="col2">
          <s:radio list="#{1:'男',0:'女'}" label="性 别" name="gender" value=
           "1"></s:radio>
        </div>
      <div class="clear"/>
          <div class="col1" style="width:400px;">
              请选择头像
          </div>
          <div class="col2">
            <s:radio list="#{'1.gif':'<img src=images/head/1.gif >',
              '2.gif':'<img src=images/head/2.gif>',
              '3.gif':'<img src=images/head/3.gif>',
              '4.gif':'<img src=images/head/4.gif>'
            }"name="head" value="'1.gif'"></s:radio>
          </div>
          <div class="clear"/>
          <div class="col1"></div>
          <div class="col2">
              <s:submit cssClass="btn" value="注 册" id="submit"></s:submit>
          </div>
      </s:form>
  </BODY>
  </HTML>
```

在代码 13-2 中，一旦发现某个表单项的信息输入不符合程序约束，就利用 JQuery 在该表单项之后添加一个 div 标签，显示错误信息提示，同时利用一个变量 error 对错误设计

数。当 error 不为 0 时，submit 的 click 事件方法返回 false，从而阻止浏览器将请求提交给 Web 服务器。

代码 13-2 在浏览器中的运行情况如图 13-2 所示。



图 13-2　利用 JQuery 实现注册信息的客户端验证

13.3　利用 JQuery 实现 AJAX

13.3.1　JSON

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式。在 AJAX 开发时，通常利用 JSON 作为 Web 服务器和 JavaScript 之间交换数据的格式。

JSON 支持两种结构。

(1) “名称/值”对的集合，可以理解成高级语言中的对象(object)、记录(record)、结构(struct)、哈希表(hash table)和有键列表(keyed list)等。在这种结构中，对象是一个无序的“name/value”集合。对象以“{”(左括号)开始，“}”(右括号)结束。每个 name 后跟一个“:”(冒号)。两个“name/value”之间使用“,”(逗号)分隔。例如：

```
{"userName":"zhangsan","email":"zhangsan@163.com","age":30, "marriage":false}
```

这里的 value 也可以是一个 Object 或者数组。例如，某个 Person 对象包括 UserName 和 address 两个对象，而 address 又由对象 city、street 和 ZIP 构成，可以写成如下形式。

```
{"userName":"zhangsan","address":{"city":"BeiJing","street":"NanJing Road","ZIP":100000}}
```

(2) 值的有序列表，可以理解为高级语言中的数组(array)。当需要表示一组值时，JSON 不但能够提高可读性，而且可以减少复杂性。例如，对于一个名为 student 的变量，值是包含三个条目的数组，每个条目是一个人的记录，包含姓名、年龄和 E-mail 地址，

可以写成如下形式。

```
{ "student": [
    { "name": " zhangsan ", "age":21, "email": "zhangsan@163.com" },
    { "name": "lisi", "age":20, "email": "lisi@sina.com" },
    { "name": "wangwu", "age":19, "email": "wang@hotmail.com" }
]}
```

13.3.2 JQuery 的 AJAX 方法

除了完成 DOM 操作和输入验证外，JQuery 还提供了一套完整的 AJAX 函数和方法，使得程序员能够很方便地开发 AJAX 应用。表 13-5 给出了所有 JQuery 提供的 AJAX 函数和方法。

表 13-5 AJAX 函数和方法

函 数	说 明
jQuery. ajax()	执行异步 HTTP(AJAX)请求
jQuery. ajaxSetup()	设置将来的 AJAX 请求的默认值
jQuery. get()	使用 HTTP GET 请求从服务器加载数据
jQuery. getJSON()	使用 HTTP GET 请求从服务器加载 JSON 编码数据
jQuery. getScript()	使用 HTTP GET 请求从服务器加载 JavaScript 文件，然后执行该文件
. load()	从服务器加载数据，然后返回到 HTML 放入匹配元素
jQuery. param()	创建数组或对象的序列化表示，适合在 URL 查询字符串或 AJAX 请求中使用
jQuery. post()	使用 HTTP POST 请求从服务器加载数据
. serialize()	将表单内容序列化为字符串
. serializeArray()	序列化表单元素，返回 JSON 数据结构数据

除上述方法外，JQuery 还提供了一些 AJAX 事件，允许程序员在 AJAX 处理前后做一些附加的操作，例如捕捉 AJAX 错误、AJAX 请求成功提示等，如表 13-6 所示。

表 13-6 AJAX 事件

事 件	说 明
. ajaxComplete()	当 AJAX 请求完成时，注册要调用的处理程序
. ajaxError()	当 AJAX 请求完成且出现错误时，注册要调用的处理程序
. ajaxSend()	在 AJAX 请求发送之前显示一条消息
. ajaxStart()	当首个 AJAX 请求完成开始时，注册要调用的处理程序
. ajaxStop()	当所有 AJAX 请求完成时，注册要调用的处理程序
. ajaxSuccess()	当 AJAX 请求成功完成时，显示一条消息

下面举一个例子，演示利用 JQuery 的 load()方法，将服务器上的 load. html 文件动态加载到当前页面的 div 中，如代码 13-3 所示。

代码 13-3 load()方法的用法

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>load 函数</title>
<script src="js/jquery-1.8.1.min.js"></script>
<script>
    $(function(){
        $("#expand").click(function(){
            $("#content").load("load.html", function(){
                $("#content").fadeIn('slow');
            });
        });
    })
</script>
</head>
<body>
    <input type="button" id="expand" value="expand"/><br/>
    <div id="content"></div>
</body>
</html>
```

上述代码在 Chrome 5 浏览器中无法加载 load.html，因为 Chrome 5 对非针对服务端的 AJAX 调用做了严格的限制。代码 13-3 在 Firefox 浏览器中的运行效果如图 13-3 所示，图中显示为单击 expand 按钮后的页面。

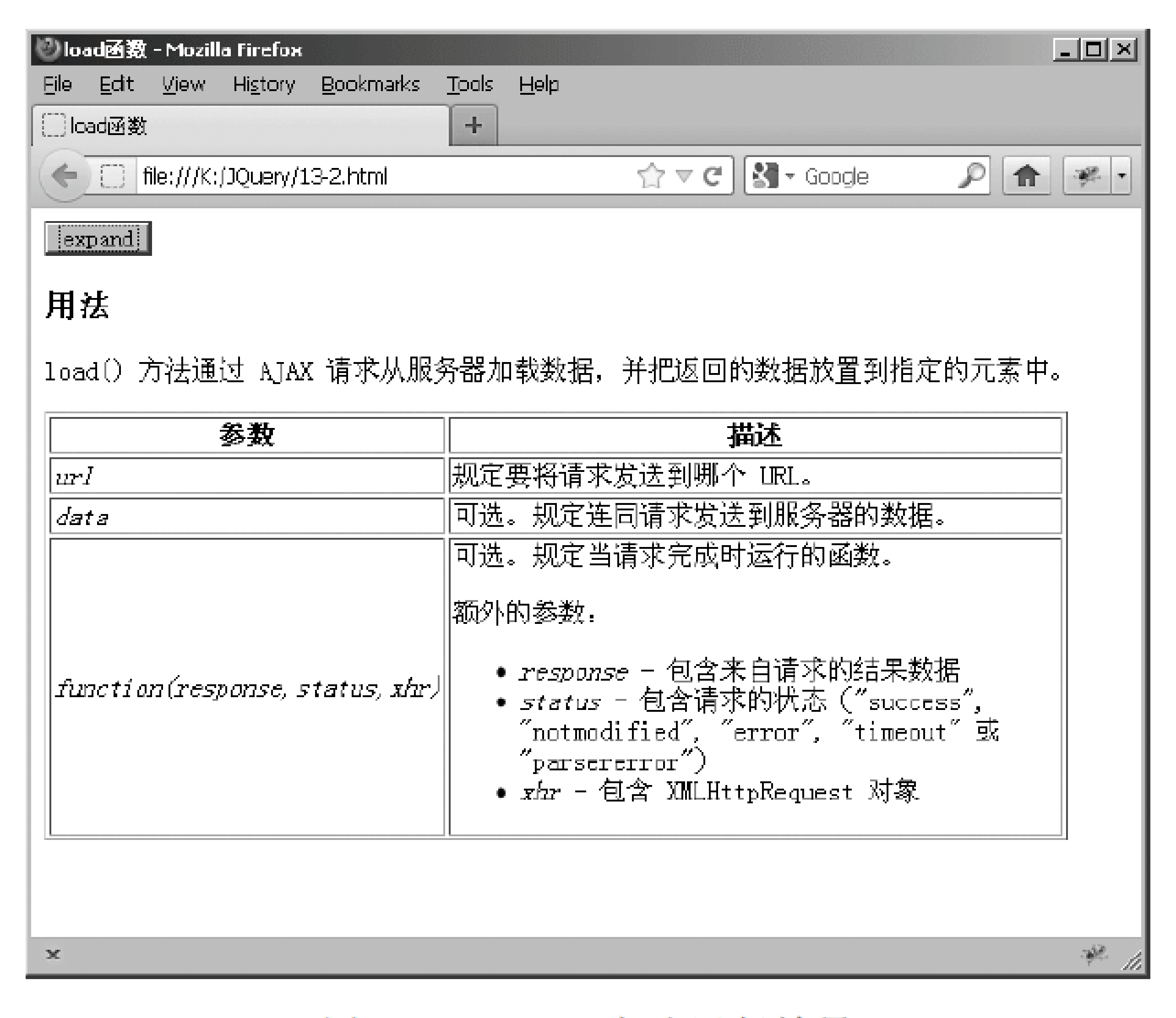


图 13-3　load()方法运行效果

load()方法只能从服务器上获取静态文件。如果希望获取动态数据，可以使用 get()或 post()方法。

get()方法的结构如下：

```
$ (selector).get(url,data,function(response,status,xhr),dataType)
```

其中,dataType为务器响应的数据类型,可以为xml、html、text和JSON等数据类型,其余各参数的含义与load()方法相同。

post()方法的结构和使用方式与get()方法是一样的,分别对应HTTP的GET操作和POST操作,在此不再赘述。

13.3.3 调用Action返回JSON字符串

在基于Struts 2的Web开发中,可以利用JQuery调用Action类完成某项处理,并将处理的结果利用JSON返回给页面。

下面通过一个例子详细讨论如何实现Struts 2、JQuery和JSON的整合。该例子演示了在Web应用程序注册新用户时,输入用户名后,浏览器会利用AJAX技术查询该用户名是否已经存在。

1. 整合Struts 2、JQuery和JSON时所需要的JAR

在整合Struts 2、JQuery和JSON时,除了需要导入4.2.1小节提到的7个基本JAR包外,还需要导入json_simple-x.x.jar和Struts 2-json-plugin-x.x.x.jar。

json_simple-x.x.jar是一个简单的Java类库,用于解析和生成JSON文本。该类库提供的JSONValue、JSONObject、JSONArray等类,为在Java应用程序中处理各种数据类型JSON的数据提供了便利。由于json_simple-x.x.jar在实现时不依赖于其他类库,因此性能比较高。

Struts 2-json-plugin-x.x.x.jar是Struts 2提供的JSON插件,定义了用于响应客户端请求的JSONResult结果类型,以及读写JSON的JSONReader和JSONWriter等类。

2. 编写Action类

代码13-4给出了检查用户名是否已经存在的Action类,它与普通的Action类基本类似,不同的是在该类的execute()等方法中通常只需要返回SUCCESS结果码。该类提供了一个result属性,用于向客户端返回一个JSON。由于result仅仅返回一个字符串,所以只需提供getter方法即可,不需要额外的处理。

代码13-4 用于返回JSON数据的Action类

```
package notes.action;
import notes.dao.UserDao;
import notes.dao.impl.UserDaoImpl;
import com.opensymphony.xwork2.ActionSupport;
public class ValidUserAction extends ActionSupport{
    private String userName;
    private String result;              //用于返回结果
    //省略了 getter 和 setter 方法

    public String execute() throws Exception {
        UserDao userDao= new UserDaoImpl();
```

```
            if(userDao.findUser(userName)==null)
                result="用户名可用!";
            else
                result="用户名已经存在!";
            return SUCCESS;
        }
    }
```

3. 编写 JS 脚本

JQuery 中常用 getJSON 来调用并获取远程的 JSON 字符串，将其转换为 JSON 对象。如果成功，则执行回调函数。

代码 13-5 在 userName 输入框的 blur()方法中，通过 getJSON()方法调用了服务器端的 validUser. action 实例，同时将用户名传递给 Action 类的 userName 属性，并在成功获得响应后，将返回的 JSON 数据（存放在 result 中）添加到一个 div 中。请注意 getJSON()方法的第二个参数，该参数为发送到服务器端的数据，其格式必须是 JSON 支持的两种格式之一。当需要将表单中输入的所有数据都发送到服务器端时，也可以直接使用 JQuery 提供的序列化函数，形如 $("#form1").serialize()。

代码 13-5　通过 JQuery 调用 Action 类的 JS 文件

```
$(function(){
    $("#userName").blur(function(){
        $.getJSON("validUser.action",           //后台处理程序
            {userName:$("#userName").val()},    //要传递的数据
            function(result){                   //回调函数
            var tip="<div style='color:red'id='userTip'>"+result+"</div>"
            if(!exist("userTip"))                   /*若不存在 userTip 的 div,则添加一个*/
                $("#userName").after(tip);
            else{                                   /*存在 userTip 对象,直接显示*/
                $("#userTip").html(tip);
                $("#userTip").show();
            }
        });
    })
});
```

4. 编写 struts.xml

在整合 JSON 时，所有返回 JSON 数据的 Action 类必须配置在扩展自 json-default 的 package 中。另外，Struts 2 提供的 JSON 插件 Struts 2-json-plugin-x. x. x. jar 中定义了一个用于响应客户端请求的 json 结果类型，必须为 Action 类中配置该结果类型。

代码 13-6 给出了 ValideUser. action 的配置，注意，JSON 返回类型中的参数 root 指出了用户返回 JSON 数据的 Action 属性。

代码 13-6　struts. xml

```
<struts>
    <package name="default" namespace="/" extends="json-default">
        <action name="validUser" class="notes.action.ValidUserAction" >
            <result type="json">
                <param name="root">result</param>
            </result>
        </action>
    </package>
</struts>
```

13.3.4 调用 Action 返回 List

在整合 Struts 2、JQuery 和 JSON 时，除了通过 Action 类返回一个字符串，还可以返回一个 List。

下面举一个例子，利用 JQuery 调用 Action 类获取所有的留言信息。

1. 编写 Action 类

代码 13-7 给出了返回所有留言信息的 Action 类。

代码 13-7 ListNotes. Action 类

```
package notes.action;
import java.util.List;
import notes.dao.* ;
import notes.model.Notes;
import com.opensymphony.xwork2.ActionSupport;

public class ListNotesAction extends ActionSupport {
    private List<Notes> notes;
    //省略了 getter 和 setter 方法

    public String execute() {
        NotesDao notesDao=new NotesDaoImpl();
        notes=notesDao.getAllNotes();
        return SUCCESS;
    }
}
```

由上述代码可以看出，在集成 JQuery 和 Struts 2 时，返回 List 的 Action 类的写法与以前学到的内容没有区别。

ListNotes. Action 类在 struts. xml 中配置的方法与 13. 3. 3 小节类似，读者可仿照代码 13-6 操作。

2. 编写页面文件 listNotes.jsp

本节将使用 JQuery 的 ajax()方法完成对 Action 类的访问。ajax()方法是最底层的 AJAX 实现，通常情况下，用户只需使用前面提到的 get()、post()、load()、getJSON()和 getScript()等方法完成 AJAX 操作，但是有时出于灵活性考虑，也会用到 ajax()方法。

ajax()方法只有一个可选的参数，该参数的值是一个“名称/值”对的集合，包含了所需要的请求设置以及回调函数等。集合中的每一个设置参数都是可选的。表 13-7 给出了常用的设置参数。

表 13-7　ajax()方法的参数中常用的设置

事　件	类　型	说　　明
url	String	用于处理请求的地址。默认值为当前页面的地址
type	String	请求方式，可选值 post 和 get。默认值为 post
data	Object 或 String	发送到服务器的数据，将自动转换为请求字符串格式。GET 请求中将附加在 URL 后。必须为 Key/Value 格式。如果为数组，JQuery 将自动为不同值对应同一个名称。例如，{foo：["bar1"，"bar2"]} 转换为 '&foo=bar1&foo=bar2'
dataType	String	期望服务器返回的数据类型。可用值为： • xml：返回 XML 文档，可用 JQuery 处理 • html：返回纯文本 HTML 信息；包含的 Script 标签会在插入 DOM 时执行 • script：返回纯文本 JavaScript 代码。不会自动缓存结果，除非设置了 "cache"参数。注意：在远程请求时(不在同一个域下)，所有 POST 请求都将转为 GET 请求 • json：返回 JSON 数据 • jsonp：JSONP 格式。使用 JSONP 形式调用函数时，如"myurl?callback=?"，JQuery 将自动替换“?”为正确的函数名，以执行回调函数 • text：返回纯文本字符串 缺省时，JQuery 会自动根据 HTTP 包 MIME 信息来智能识别
beforeSend	Function	在发送请求前执行，可修改 XMLHttpRequest 对象的函数。有唯一的参数：XMLHttpRequest 对象
complete	Function	请求完成后的回调函数(请求成功或失败之后均调用)。有唯一的参数：XMLHttpRequest 对象
success	Function	请求成功完成后的回调函数，包括两个可选参数： • 服务器返回的数据，类型取决于 dataType 设置 • 描述状态的字符串
error	Function	请求失败后的回调函数，包括三个参数：XMLHttpRequest 对象、错误对象和捕获错误的对象
global	Boolean	是否触发全局 AJAX 事件。默认值为 true

在代码 13-8 中给出的页面文件利用 ajax()方法实现了对 Action 类的调用。

代码 13-8　页面文件 listNotes.jsp

```
<%@page language="java" pageEncoding="UTF-8"%>
<html>
<head>
<title>留言板--列表</title>
    <script type="text/javascript" src="js/jquery-1.8.1.js"></script>
<script language="javascript">
    $(function(){
        $.ajax({
```

```
                type:"post",                    //post 方式
                dataType:"json",                //返回数据类型为 JSON 数据格式
                url:"listNotes.action",         //所调用的 Action 名称
                success:function(data){
                //使用 jQuery 中的 each(data,function(){});函数
                //从 data 获取 Notes 对象放入 note 之中
                $.each(data,function(i,note){
                    $("#message").append("<div>第"+(i+1)+"条留言:</div>")
                    .append("<div style='color:red'>留言人:
                        "+note.user.userName+"</div>")
                    .append("<div style='color:red'>主题:"+note.title+"</div>")
                    .append("<div style='color:red'>内容:"+note.content+"</div>");
                    });
                }
                });
        })
    </script>
    </head>
    <body>
        <div id="message"></div>
    </body>
    </html>
```

代码 13-8 中，在 ajax()的 success 函数中使用 each()方法对接收到的 List 进行了遍历。图 13-4 给出了 listNotes.jsp 页面的运行情况。



图 13-4 listNotes.jsp 运行效果

同 步 训 练

修改站内短信，利用 JQuery 实现下述功能。

(1) 添加短信删除功能。

(2) 为编写短信添加保存到草稿箱的功能。

Appendix A 附录A MyEclipse 常用的快捷键

1. 编辑常用快捷键

Ctrl+S	保存快捷键
Ctrl+D	删除当前行
Ctrl+Shift+F	格式化代码
Ctrl+/	注释或者取消注释选中行(或鼠标所在行)
Ctrl+Shift+X	所选字符转换为大写
Ctrl+Shift+Y	所选字符转换为小写
Ctrl+Shift+↓	复制当前行
Ctrl+Shift+P	查找与当前内容相匹配的内容(例如括号的匹配)

2. 内容提示快捷键

Ctrl+1	提示如何修正错误,可实现重命名
Ctrl+Shift+O	自动导入所有未导入的包或类
Ctrl+Shift+M	导入光标所在行的未导入的类或包
Alt+/	内容辅助,自动补全
Ctrl+T(F4)	显示类结构
Ctrl+O(F3)	显示类中方法和属性大纲

3. 重构快捷键

Alt+Shift+R	重命名
Alt+Shift+M	抽取方法
Alt+Shift+C	修改函数结构
Alt+Shift+L	抽取本地变量
Alt+Shift+F	把 Class 中的 local 变量变为 field 变量
Alt+Shift+I	合并变量(可能这样说有点不妥 Inline)
Alt+Shift+V	移动函数和变量
Alt+Shift+Z	撤销重构



EL 表达式

1. EL表达式简介

EL(Expression Language)提供了在 JSP 中简化表达式的方法,它允许在脚本编制元素范围外使用运行时表达式。所谓的脚本编制元素,是指页面中能够用于在 JSP 文件中嵌入 Java 代码的元素。它们通常用于对象操作,以及执行那些影响所生成内容的计算。利用 EL 表达式可以完成如下功能。

(1) 获取数据:EL 表达式通常用来替换页面中的 JSP 表达式,可以将各种类型的 Java 对象的值输出到 JSP 页面上。

(2) 执行运算:利用 EL 表达式,可以在 JSP 页面中执行一些基本的关系运算、逻辑运算和算术运算,以在 JSP 页面中完成简单的逻辑运算。

(3) 获取 Web 开发常用对象:EL 表达式语言中定义了 11 个隐含对象,使用这些隐含对象可以很方便地获取 Web 开发中的常见对象,并读取这些对象的数据。

(4) 调用 Java 方法:在 JSP 页面中,通过 EL 表达式可以调用 Java 类的方法。

2. EL语法规则

调用 EL 表达式的一般格式如下:

```
${expression}
```

注意:$ 和{}不要漏写,它是组成 EL 表达式不可缺少的一部分。

EL 表达式语句在执行时会调用 pageContext.findAttribute()方法,用标识符为关键字,将按照 page→request→session→application 的顺序查找相应的对象。找到,则返回相应对象;找不到,则返回""。

利用 EL 表达式还可以很方便地获取 JavaBean 的属性,或者获取数组、Collection、Map 类型集合的数据,例如:

```
${notes.user.userName}
${notes.list[0]}              //访问有序集合某个位置的元素
${map.key}                    //获得 map 集合中指定 key 的值
```

3. []与.运算符

EL 提供“.”和“[]”两种运算符来存取数据。

当要存取的属性名称中包含一些特殊字符,如“.”或“?”等并非字母或数字的符号,

一定要使用“[]”。例如，${user.My-Name}应当改为${user["My-Name"]}。

如果要动态取值，可以用“[]”来完成，而“.”无法做到动态取值。例如，${sessionScope.user[data]}中的 data 是一个变量。

4. EL 表达式中的隐含对象

EL 表达式提供了 11 个隐含对象，用于访问 JSP 中的常用对象，如表 B-1 所示。访问隐含对象的语法为：

```
${隐式对象名称}
```

表 B-1　EL 表达式中的隐含对象

隐含对象名称	描　述
pageContext	表示当前页面的 Javax.servlet.jsp.PageContext 对象
pageScope	表示 page 域中用于保存属性的 Map 对象
requestScope	表示 request 域中用于保存属性的 Map 对象
sessionScope	表示 session 域中用于保存属性的 Map 对象
applicationScope	表示 application 域中用于保存属性的 Map 对象
param	表示保存了所有请求参数的 Map 对象
paramValues	表示保存了所有请求参数的 Map 对象。对于某个请求参数，返回的是一个 string[]
header	表示保存了所有 http 请求头字段的 Map 对象
headerValues	表示保存了所有 http 请求头字段的 Map 对象，返回 string[]数组
cookie	表示保存了所有 cookie 的 Map 对象
initParam	表示保存了所有 web 应用初始化参数的 Map 对象

5. 使用 EL 调用 Java 方法

EL 表达式语法允许程序员开发自定义函数，以调用 Java 类的方法。格式如下：

```
${prefix:method(params)}
```

注意：在 EL 表达式中调用的只能是 Java 类的静态方法，并且这个 Java 类的静态方法需要在 TLD 文件中描述，才可以被 EL 表达式调用。

下面给出一个自定义 EL 函数的例子，Java 代码如下：

```
import Java.io.UnsupportedEncodingException;
import Java.net.URLEncoder;
public class MyELTag {
    public static String myString(String str) {
        String decodeStr="["+ str+ "]";
        return decodeStr;
    }
}
```

为了在 EL 表达式中调用该函数，需要首先为其配置 TLD 文件。为此，在 WebRoot\WEB-INF 目录下面建立一个 myTag.tld 文件，内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://Java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://Java.sun.com/xml/ns/j2ee http://Java.sun.com/xml/ns/j2ee/web-
                         jsptaglibrary_2_0.xsd"
        version="2.0">
        <tlib-version>1.0</tlib-version>
        <short-name>el</short-name>
<function>
<!--对这个 EL 方法的描述 -->
        <description>calculate string length</description>
        <name>FunctionsEl</name><!--调用 EL 方法的名称 -->
            <function-class>com.el.code.FunctionsEl</function-class>
            <function-signature>
                Java.lang.String elEncode(Java.lang.String)
            </function-signature>
        <example>${el:FunctionsEl(str)}</example><!--例如 -->
</function>
</taglib>
```

以后就可以在 JSP 页面采用下面的方式使用 EL 表达式引用自定义函数。

```
<%@taglib prefix="el" uri="/WEB-INF/myTag.tld"%>
${el:myString("Hello World!")}
```

6. EL 表达式与 Struts 2

尽管在 Struts 2 框架中将 application 和 session 等对象进行了包装，仍然可以利用 EL 表达式来完成对这些对象的存取。

例如，在一个 Action 类的 execute()方法中具有如下代码。

```
ActionContext context=ActionContext.getContext();
Map session=context.getSession();
Map application=context.getApplication();
session.put("user", "zhangsan");
application.put("welcome", "欢迎登录留言板");
message="Hello World!";          //message 是 Action 类的属性
```

则在对应的 JSP 页面中，可以使用下述代码完成对 session、application 和 Action 类的属性的存取。

```
Session:${session.user} <br/>
Application:${application.user2 } <br/>
message:${message }<br/>
who:${who}
```

参 考 文 献

[1] 赵会东. ASP. NET 开发宝典[M]. 北京:机械工业出版社,2012.
[2] 耿祥义,张跃平. JSP 实用教程[M]. 北京:清华大学出版社,2003.
[3] 林振荣,徐苏. JSP 程序设计[M]. 北京:中国铁道出版社,2008.
[4] 陈臣. 研磨 Struts 2[M]. 北京:中国铁道出版社,2008.
[5] 孙鑫. Struts 2 深入详解[M]. 北京:电子工业出版社,2008.
[6] 李刚. Struts 2 权威指南[M]. 北京:电子工业出版社,2007.
[7] Kurniawan B.. 深入浅出 Struts 2[M]. 杨涛,王建桥,杨晓云,译. 北京:人民邮电出版社,2009.
[8] 单东林,张晓菲,魏然. 锋利的 JQuery[M]. 北京:人民邮电出版社,2009.